МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

О. І. Пушкар
Є. М. Грабовський

Проєктування додатків для мобільних пристроїв

Навчальний посібник

Харків
ХНЕУ ім. С. Кузнеця
2023

УДК 004.4(075.034)
П91

Авторський колектив: д-р екон. наук, професор О. І. Пушкар – підрозд. 1, 4, 6; канд. екон. наук, доцент Є. М. Грабовський – вступ, підрозд. 2, 3, 5.

Рецензенти: начальник кафедри військового зв'язку та інформатизації Національної академії Національної гвардії України, полковник, д-р техн. наук, професор *О. Ю. Іохов*; завідувач кафедри технології поліграфічного виробництва Видавничо-поліграфічного інституту Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", д-р техн. наук, доцент *О. О. Палюх*.

Рекомендовано до видання рішенням ученої ради Харківського національного економічного університету імені Семена Кузнеця.
Протокол № 9 від 20.12.2022 р.

Самостійне електронне текстове мережеве видання

Пушкар О. І.

П91 Проєктування додатків для мобільних пристроїв [Електронний ресурс] : навчальний посібник / О. І. Пушкар, Є. М. Грабовський. – Харків : ХНЕУ ім. С. Кузнеця, 2023. – 167 с.

ISBN 978-966-676-858-5

Розглянуто ключові особливості проєктування мобільних застосунків на платформі операційної системи Android. Надано інформацію про основи програмування мобільних застосунків.

Рекомендовано для студентів спеціальності 186 "Видавництво та поліграфія" другого (магістерського) рівня усіх форм навчання.

УДК 004.4(075.034)

ISBN 978-966-676-858-5

Вступ

Значна поширеність сучасних ґаджетів обумовлює необхідність у створенні відповідного програмного, інформаційного та технічного забезпечення під ці пристрої. Інформаційним та програмним забезпеченням мобільних пристроїв є різноманітні застосунки (додатки), які виконують певні функціональні завдання. Нині операційна система Android є найпопулярнішою платформою для мобільних пристроїв. Різноманітність і значне поширення смартфонів і планшетів різних виробників, що функціонують під управлінням цієї платформи, стимулює зростання ринку мобільних застосунків (додатків), роблячи навички в розробленні під Android дуже затребуваними на сьогодні. Саме тому актуального значення набуває вивчення особливостей створення застосунків для мобільних пристроїв під управлінням операційної системи Android.

Навчальна дисципліна "Проєктування додатків для мобільних пристроїв" є обов'язковою навчальною дисципліною, що вивчають, згідно з навчальним планом підготовки фахівців освітньо-кваліфікаційного рівня "магістр" спеціальності 186 "Видавництво та поліграфія" для всіх форм навчання.

Метою вивчення навчальної дисципліни є надання здобувачам вищої освіти сучасних теоретичних знань загальних особливостей мобільних застосунків, а також формування у студентів відповідних компетентностей щодо створення застосунків для мобільних пристроїв під управлінням операційної системи Android.

Об'єктом навчальної дисципліни є процес розроблення застосунків для мобільних пристроїв.

Предметом навчальної дисципліни є інструментальні засоби розроблення **застосунків** для мобільних пристроїв.

Завданнями навчальної дисципліни "Проєктування додатків для мобільних пристроїв" є такі:

аналіз поняття та видів мобільних застосунків;

дослідження архітектури мобільних застосунків;

вивчення особливостей візуалізації інформації для використання;

в мультимедійних застосунках;

опанування основ розроблення програм для операційної системи Android;

дослідження програмних засобів роботи з ресурсами;

аналіз механізмів зберігання даних.

У результаті вивчення навчальної дисципліни студент має набути таких **компетентностей**:

здатність застосовувати знання у практичних ситуаціях;

здатність спілкуватися іноземною мовою;

здатність оцінювати та забезпечувати якість виконуваних робіт;

здатність організовувати діяльність та ефективно управляти установами/підрозділами у сфері видавництва й поліграфії;

здатність застосовувати сучасні методи й інструменти для досліджень у сфері видавництва та поліграфії, а також забезпечувати якість продукції;

здатність аналізувати структуру та контент проєктів інтерактивних медіа.

Згідно з робочою програмою **навчальної дисципліни** "Проєктування додатків для мобільних пристроїв" цей навчальний посібник складається із двох розділів – "Загальні особливості проєктування мобільних **застосунків**" та "Основи програмування мобільних **застосунків**".

Темами навчального посібника є такі: "Поняття та види мобільних застосунків", "Архітектура мобільних застосунків", "Особливості візуалізації інформації для використання в мультимедійних застосунках", "Основи розроблення програм для операційної системи Android", "Робота з ресурсами в операційній системі Android", "Зберігання даних на платформі Android".

Тему "Особливості візуалізації інформації для використання в мультимедійних застосунках" розроблено на основі наукових результатів досліджень автора, наведених у статті [6].

До кожної теми навчального посібника наведено практичну складову, яка містить методичні вказівки для виконання лабораторних робіт із відповідної теми.

Розділ 1
Загальні особливості проєктування мобільних застосунків

1. Поняття та види мобільних застосунків

Мета – дослідження поняття, видів та функцій мобільних застосунків.

Професійні компетентності: здатність організовувати діяльність та ефективно управляти установами/підрозділами у сфері видавництва й поліграфії; здатність аналізувати структуру та контент проєктів інтерактивних медіа.

1.1. Поняття та загальні особливості мобільних застосунків

 **Мобільний застосунок** – це спеціально розроблене під функціональні можливості ґаджетів програмне забезпечення (ПЗ) [1]. Призначення ПЗ може бути різноманітним: сервіси, магазини, розваги, онлайн-помічники та ін. Ці програми завантажують та встановлюють самі користувачі через мобільні маркетплейси. Найбільші майданчики – це AppStore, Google Play. Технічно всі програми створюють під конкретну платформу мобільного ґаджета. Найбільш популярні операційні системи – це iOS, Android, Windows Phone.

Досить часто користувачі плутаються у функціональних відмінностях мобільного сайта та програми на смартфон, планшет або інший ґаджет. Буває й так, що стартапи й навіть вже розкручені бізнеси не розуміють, навіщо платити за розроблення окремого програмного забезпечення для телефону, коли можна підлаштувати під дозвіл мобільного дисплея готовий сайт.

Розбираймося у ключових відмінностях і функціональних особливостях адаптованого сайта та програми.

Мобільний варіант сайта є переробленим, а в деяких варіантах адаптованим дизайном і контентом вебсторінок для зручного перегляду на дисплеї смартфона. Найпростіший спосіб – створити копію основного сайта для ПК і спробувати його підлаштувати під мобільний дозвіл. Оптимальний варіант – це повністю "перебрати" сайт і створити новий дизайн, із яким зручно буде взаємодіяти користувачеві за допомогою сенсорного екрана.

Відповідно, просте пристосування настільної версії під ґаджети називають "гумовим" верстанням. Тобто на сайті залишається той самий контент і дизайн, але він змінюється в розмірах. Блоки стають меншими. Таке рішення було найпопулярнішим 10 – 15 років тому, коли продажі з мобільних ґаджетів не могли конкурувати з персональним комп'ютерами. Зараз багато що змінилося. Із мобільного каналу йде більше трафіка та продажів. Тому "гумовий" сайт поступається в лідогенерації мобільним застосункам.

Основні переваги сайта – мінімальні витрати на розроблення (насправді, версію для смартфонів зроблено "на додаток" з основного сайта), кросплатформеність, швидкі оновлення.

Один істотний недолік перекреслює всі переваги. Адаптований сайт має низький рівень взаємодії з користувачем. Навіть за якісного трафіка такий сайт мало буде конвертувати лідів.

 **Мобільний застосунок** – це програмний пакет, функціонал і дизайн якого "заточено" під можливості мобільних платформ.

Перелічимо кілька основних плюсів програми [3]:

інтерфейс програми створено безпосередньо під роботу на мобільному пристрої через сенсорний екран чи кнопки;

зручна та зрозуміла для користувачів ґаджетів навігація, мобільне меню;

найкраща взаємодія з користувачем через повідомлення, пуш-повідомлення, нагадування. Програма може виконувати функції навіть

у фоновому режимі, чого не можна сказати про сайт. Для роботи із програмою не потрібно відкривати браузер, а багато програм підтримують ряд функцій і в разі вимкненого інтернету;

зберігання персональних даних користувача. Ця функція розширює можливості персоналізації програм. Наприклад, викликає таксі додому (прописка), записує на приймання до лікаря з медичного поліса та інші переваги;

гнучкіший зворотний зв'язок із компанією, сервісом;

можна використовувати більше ресурсів. Наприклад, увімкнути геолокацію та викликати машину в будь-яку точку міста;

програми можуть ураховувати біологічні ритми людини та сповіщати її про необхідність дотримуватися режиму.

Насправді функціонал мобільних програм уже давно перевершив адаптовані сайти. Сьогодні можна завантажити та встановити на смартфон

програми для бізнесу, навчання, органайзери з опціями нагадування, розважальний контент, різноманітні сервісні служби.

Успіх будь-якої програми визначено не тільки новими технологіями, але й здатністю їхнього правильного застосування. Користувачам є цікавими не просто технології, їх приваблюють нові способи розв'язання наявних проблем, тому бездротові програми будуть привабливими лише в тому разі, якщо вони є корисними кінцевим користувачам.

Багато важливих програм мобільного бізнесу виникли в результаті інтеграції бездротових технологій із наявними електронними бізнес-застосунками. Одночасно унікальні властивості мобільного бізнесу – мобільність, локалізація, ідентифікація тощо – дають поштовх для розвитку нових застосунків та бізнес-моделей.

Успішний бізнес має йти в ногу із сучасними технологіями, інакше він залишиться в минулому та занепаде. Перед комерсантами мобільні програми відкривають такі перспективи:

Зростання продажів. Мати прибуток із нових каналів залучення клієнтів – це основна комерційна мета будь-якого бізнесу. Раніше мобільний трафік уважали просто одним із додаткових каналів генерації лідів. Основною платформою були сайти на комп'ютері. 2018 року експерти вважали, що 47 – 50 % продажів генерують застосунки. Конверсія з такого програмного забезпечення є в 3 – 4 рази вищою, ніж із сайтів. Це говорить про те, що продаж неминуче переходить у мобільну сферу, а значить, і бізнес має більш активно розвивати цей канал.

Висока лояльність клієнтів. Смартфон набагато частіше міститься в зоні контакту із клієнтом, ніж сайт. Користувачеві простіше замовити товар через програму, знайшовши потрібну іконку на своєму телефоні, ніж шукати сайт на комп'ютері. Компанія може стимулювати інтерес клієнта пуш-повідомленнями – це дешевий та досить ефективний варіант підвищення лояльності цільової аудиторії (ЦА).

Автоматизація частини бізнес-процесів. Наприклад, виклик таксі через програми дозволяє розвантажити диспетчерські служби. Також у ресторанах і кафе можна замовляти їжу, бронювати квитки на рейси та багато іншого. Для малого та середнього бізнесу випускають програми, які можуть приймати й опрацьовувати замовлення, переспрямовувати виклики на фахівців.

Приймання платежів та співпраця з онлайн-транзакціями. Сьогодні багато хто користується мобільними застосунками для інтернет-банкінгу, електронними грошима та сервісами кешбек. Компанії можуть легко

налаштувати приймання платежів на своїй програмі, під'єднавшись до одного із провідних агрегаторів.

Аналіз цільової аудиторії (ЦА). За допомогою застосунків можна здобути ряд додаткових метрів поведінок про цільову аудиторію сайта, компанії, продукту. Сьогодні аналітичні системи можуть розподіляти трафік по пристроях входу. Така інформація допоможе створити більш точний портрет ЦА, а отже, розробити ефективні маркетингові програми.

Скорочення витрат на утримання штату працівників. Наприклад, деякі програми для бізнесу можуть розв'язати для компанії проблему пошуку операторів кол-центру. Крім того, використовуючи пуш-повідомлення можна в рази скоротити бюджет на контекстну рекламу та email-маркетинг.

Підтримання користувачів, сервісна служба. Мобільні програми можуть бути центром підтримання користувачів. Основне завдання таких програм – ефективне спілкування із клієнтами. Наприклад, програма може допомогти налаштувати платежі, автооплату, заявку на виклик спеціаліста, переспрямувати на чат зі співробітником та інше.

Рейтинг популярності мобільних застосунків станом на кінець 2021 року показано на рис. 1.1.

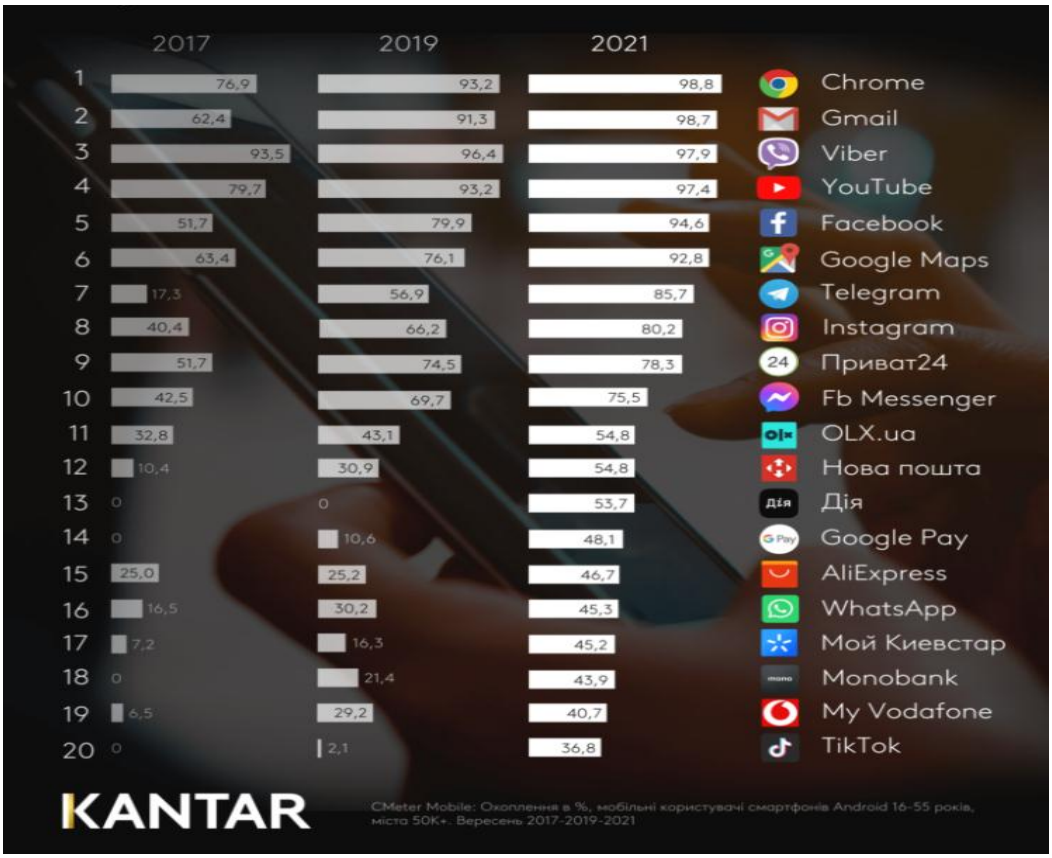


Рис. 1.1. **Рейтинг популярності мобільних застосунків станом на кінець 2021 року** [13]

⚠ За інформацією Sensor Tower, користувачі в І кварталі 2022 року зробили більш ніж 36,9 млрд завантажень застосунків. Це на 1,4 % більше, ніж за перший квартал 2021 року (рис. 1.2).

Rank	App	Category
1	TikTok	Entertainment
2	Instagram	Photo and video
3	Facebook	Social networking
4	WhatsApp	Messaging
5	Shopee	Shopping
6	Telegram	Messaging
7	Snapchat	Photo and video
8	Messenger	Messaging
9	CapCut	Photo and video
10	Spotify	Music

Рис. 1.2. **Рейтинг найбільш завантажуваних мобільних застосунків у І кварталі 2022 року** [13]

Слід зазначити, що найбільш популярними виявилися десять найбільш завантажуваних застосунків, які становлять собою платформи соціальних мереж. Із них платформам Meta і ByteDance належать шість із десяти найкращих.

1.2. Види мобільних застосунків

⚠ За бізнес-спрямованістю мобільні програми можуть бути двох видів. Перші оптимізують внутрішні процеси організації, підприємства, сервісу. Другі використовують у маркетинговому плані, тобто для комунікацій із клієнтами, продажу та просування бренда [1].

Програми для клієнтів. Різні онлайн-сервіси, реалізовані в мобільному середовищі. До цієї групи входять програми для інтернет-банкінгу, відстеження посилок, бронювання квитків і номерів у готелях, різні маркетплейси, онлайн-вітрини з товарами та послугами.

Програми лояльності клієнтам. Програми на кшталт знижкових і бонусних карток для постійних клієнтів, дисконтні програми, кешбек та ін.

Програми для внутрішнього використання. Мають на увазі програми, які оптимізують колективну роботу та спілкування. Це різноманітні месенджери, хмарні сховища, віртуальні офіси та ін.

Програми для автоматизації бізнес-процесів. Наприклад, автоматизація замовлень у ресторанах, купівель у торговельних центрах, бронювання номерів у готелях.

Також мобільні програми можна розподілити на три типи:

*Мобільні вебзастосунки та сайти. Я*к вже зазначено раніше, у таких рішень є кілька плюсів – це кросплатформеність, простота створення й оновлення. Мінус у низькій функціональності. Це непоганий варіант для старту, щоб проаналізувати мобільний трафік у бізнес-ніші. Проте з такими програмами практично нічого не заробиш через їхній низький функціонал.

Гібридні програми – це вже більш сучасний варіант, який працює на API. У програмах вже є пуш-сповіщення, програму можна розміщувати у плейсмаркетах для вільного або платного скачування. Такі програмні рішення мають можливість незалежного оновлення, що знімає необхідність у випуску нових версій.

Нативні застосунки – це "накручені" фічі, які дають максимальну функціональність і швидкість взаємодії. Однак для їхньої стабільної роботи потрібні серйозні ресурси системи.

Мобільні програми можна також розподілити на три групи: програми для підтримання мобільних технологій (enabling applications), програми для споживачів (consumer applications) та бізнес-застосунки (business applications).

Застосунки для підтримання мобільних технологій переважно є модифікацією програм, які використовують в електронному бізнесі для управління інформацією та організаційної діяльності. Серед інших ця група застосунків містить мобільну пошту, миттєве передавання повідомлень, програми-щоденники та мобільний офіс. Програми підтримання мобільних технологій також називають горизонтальними застосунками, оскільки їх можуть використати як звичайні, так і бізнес-користувачі. Причому, якщо приватні особи використовують їх найперше для здобування останніх новин і для того, щоб підтримувати зв'язок зі своїми сім'ями та друзями, то керівним та мобільним службовцям (мобільний слуга – працівник компанії, який виконує свої службові функції поза офісом) вони потрібні передусім для здобуття доступу до внутрішніх довідників компанії, систем передавання повідомлень, списків контактів тощо.

Наступна група – це *програми для споживачів*, які становлять досить велику та важливу групу мобільних застосунків. По-перше, до цієї групи належать сервіси, які пропонують традиційний бізнес і прагнуть розширити їхню наявність і на мобільну сферу. Різні фінансові послуги, як, наприклад, виконання банківських операцій і платежів, послуги охорони, мобільні купівлі, реклама, розважальні послуги, так само як і послуги з надання мобільних новин та інформації – лише мала частина з довгого переліку подібних мобільних рішень.

 Розгляньмо докладніше третю групу застосунків, що є особливо важливою для підприємств та організацій, тобто звернімося до **мобільних бізнес-застосунків**.

Бізнес-рішення можна розподілити на застосунки для покупців, бізнес-партнерів і службовців. Застосунки для службовців, своєю чергою, може бути розподілено на декілька категорій, як-от продаж та автоматизація управління співробітниками, мобільне управління відрядженнями тощо. Багато компаній насамперед упроваджують програми для мобільних службовців, оскільки вони обіцяють швидке зростання продуктивності та зниження витрат. Крім того, такі програми зручні для проведення експериментів, моніторингу проєктів та набування необхідного досвіду.

Мобільні бізнес-застосунки є досить специфічними для процесів, що лежать в основі діяльності компанії, і тому їх часто називають вертикальними. Під час створення мобільних бізнес-рішень розробникам необхідно враховувати особливості бізнес-процесів кожної компанії. Навіть якщо процеси є дуже схожими, способи їхньої реалізації в різних компаніях можуть відрізнятися. Це означає, що мобільні програми досить складно стандартизувати. У будь-якому разі їх доводиться розробляти з нуля, ретельно описуючи схему бізнес-процесів та їхні взаємозв'язки в компанії. Навпаки, програми для підтримання мобільних технологій і програми для споживачів стандартизувати не так складно та в майбутньому можуть стати продуктом масового споживання.

1.3. Приклади мобільних застосунків для вирішення ключових завдань бізнесу

Мобільне управління взаємовідносинами із клієнтами

 Програми управління взаємовідносинами із клієнтами (Customer relationship management – CRM) призначено насамперед для поліпшення

якості тих функцій компаній, які пов'язані зі споживачами, наприклад: маркетинг, функції продажу та надання сервісів. Мобільні CRM-програми підвищують якість обслуговування клієнтів, що веде до підвищення довіри до продукту та задоволеності покупців. Традиційні CRM-рішення тут не підходять через наявний розподіл функцій працівників, що перебувають в офісі, та працівників, які взаємодіють із клієнтами поза ним. Тому інформацію не може бути повністю та швидко здобуто у вихідній базі даних, так само як її не може бути повністю продано та опрацьовано в тій точці, де надають послугу. Дотепер торговельним представникам і технічним фахівцям часто доводиться спочатку об'єднувати здобуту інформацію на папері, а вже потім уводити її до ПК. У результаті частину інформацію забувають, утрачають, а частина, що залишилася, може містити помилки. Це призводить до появи в базах даних неточної та суперечливої інформації.

Крім того, важливі відомості про споживачів, які містяться в центральній базі даних, не може бути ефективно використано працівниками, які перебувають далеко від офісу. Їм є доступною лише частина інформації. Однак висока конкуренція потребує своєчасної появи достовірної та повної інформації у всіх працівників для того, щоб побажання клієнтів виконували точніше, швидше та краще.

Для реалізації цих завдань використовують мобільні CRM-рішення, які дозволяють збирати інформацію про всі контакти із клієнтами й об'єднувати її до загального банку даних. Ці дані застосовують під час створення профілю клієнта та проведення цільових маркетингових акцій. Торговельні представники, які володіють CRM-застосунками, мають можливість здобувати інформацію про попередні контакти із клієнтом, використовувати архіви, інформацію про продаж, ціни, доставляння. Так само працівники, які перебувають поза офісом, можуть отримувати на свої мобільні пристрої технічні деталі, що їх цікавлять, іншу інформацію. Усі ці дані допомагають підвищити ефективність продажів, якість робіт, що виконують поза офісом, і, що найголовніше, поліпшити взаємодію із клієнтами.

Мобільне управління ланцюжками постачань

 Мета стратегії управління ланцюжками постачань (Supply chain management – SCM) – забезпечити безперервність фінансових, інформаційних і матеріальних потоків між постачальниками, бізнес-партнерами та споживачами.

Мобільні технології можуть суттєво поліпшити обмін інформаційними потоками між фронт- і бек-офісами компанії шляхом удосконалення механізмів збирання даних та механізмів сповіщення зацікавлених осіб про настання деяких подій. Наприклад, бездротові сканери штрих-кодів або технології розпізнавання частот радіохвиль може бути використано для збирання даних у точці появи інформації. Разом із тим зібрані дані має бути передано до центрального банку зберігання даних і в режимі реального часу поширено у всі зацікавлені підрозділи компанії. Це допомагає компанії ухвалювати більш обґрунтовані й компетентні рішення, а також поліпшувати якість планування та прогнозування. Інші можливості SCM-застосунків дозволяють управляти матеріально-технічним постачанням, запасами, внутрішньовиробничими та споживчими замовленнями, краще визначати напрями процесів, швидше передавати розпорядження, а також надають можливість поширювати накази, повідомлення, доповіді максимально зручним способом. Упровадження програм мобільного бізнесу може стати в пригоді там, де доводиться координувати роботу великої кількості людей. Такі перехресно-організаційні рішення може бути використано, наприклад, у лікарнях чи аеропортах.

Мобільна автоматизація продажів

Сьогодні торговельні представники забезпечують компаніям прибуток значною мірою завдяки мобільним технологіям, що надають їм повні відомості про покупців та товари. Раніше торговельному агентові доводилося спочатку записувати побажання клієнта на папері, а потім обговорювати їх зі службовцями для уточнення замовлення. Далі торговельний представник готував підсумкову пропозицію та надсилав її клієнту. Після здобуття підтвердження торговельний представник надсилав замовлення до відділу продажу. У разі використання мобільних технологій формування замовлення здійснюють набагато швидше, оскільки є можливість обговорити деталі з офісними службовцями в покупця будинку. Відомості про кількість запасів, дати й умови постачань, останні ціни та спеціальні акції, здобувають прямо у присутності клієнта, що значно прискорює процес укладання угоди, а також скорочує кількість претензій від клієнтів. Причому, крім здобування інформації від своєї власної компанії, торговельний агент також може оцінювати ціни та якість товарів конкурентів.

У результаті торговельним представникам легше реагувати на будь-які зміни на ринку та пропонувати клієнту максимально вигідні умови.

Мобільна автоматизація управління спільною роботою територіально віддалених підрозділів

Працівники компанії, які обслуговують клієнта поза офісом, є найбільш мобільною групою службовців компанії. Якісне обслуговування в місці продажу передбачає швидке реагування на замовлення, гнучку систему пропозицій, професійне усунення можливих помилок. Для працівників, що перебувають поза компанією, дуже важливими є бездротовий доступ до важливої інформації та можливість взаємодії з фахівцями, які перебувають у головному офісі. Крім того, роботу на виїзді завжди супроводжено великою кількістю паперових джерел інформації, що сильно знижує якість даних і часто пов'язано з виникненням помилок, що призводить до зайвих витрат. Паперова тяганина значно уповільнює весь процес угоди, роблячи його непрозорим та складним.

Можна виділити три основні переваги для службовців, які працюють із клієнтами поза офісом. По-перше, вони отримують прямо на свої мобільні пристрої поточні доручення з усіма відповідними деталями, включно з необхідними контактними даними. У результаті зникає потреба в постійних поїздках до офісу та назад. По-друге, будучи на зв'язку, службовці мають постійний доступ до інформації про попередні експлуатаційні та відновлювальні роботи, архіву послуг, а також відомостей про товари та деталі. Крім того, можна постійно мати "під рукою" величезні каталоги запасних частин, у яких нескладно знайти інформацію, необхідну для клієнта. Технічні фахівці здобувають миттєвий доступ до різноманітних інструкцій, супровідних документів та покрокових посібників.

І, нарешті, по-третє, бездротові технології надають службовцям дуже ефективні й комфортні умови для ведення звітності щодо укладання угод, витрат часу та матеріальних витрат. Подібні послуги охоплюють такі послуги, як перевірка наявності запасних деталей та їхня класифікація, заповнення форм огляду товарів і формування рахунків у режимі реального часу (online).

Телеметрія та мобільний віддалений контроль

⚠ Під "телеметрією" зазвичай розуміють "вимірювання та автоматичне передавання даних із віддалених джерел по дротах, радіо або

за допомогою інших засобів" (WordNet) [3]. Телеметрія належить до програм типу B2M (бізнес-машина), оскільки фактично передавання інформації відбувається без утручання людини. Завдяки мобільним технологіям контроль за обладнанням і механізмами можна здійснювати на відстані. Такі технології переважно використовують там, де техніка є недоступною для традиційних дротових мереж, наприклад, на гідроелектростанціях, нафтових родовищах або в інших віддалених місцях. Ще одне місце застосування віддаленого мобільного контролю – копіювальні або торговельні автомати, які можуть автономно повідомляти про необхідність у поповненні витратних матеріалів або поломки. Зазвичай такі автомати є сильно територіально розкиданими й на їхню перевірку потрібно багато часу й коштів. Віддалений мобільний контроль можна використовувати в будівлях і машинах для забезпечення їхньої безпеки. У разі виявлення проблеми сенсори просто надсилають власнику повідомлення із зазначенням місця та виду проблеми.

Допоміжні програми

До допоміжних бізнес-застосунків належить ціла низка мобільних рішень, які полегшують роботу, завдяки своєчасному передаванню необхідної інформації в потрібне місце та потрібній формі. Такі програми дозволяють особам, які ухвалюють рішення, оперативно реагувати на цю інформацію. Серед інших ця група містить мобільне управління поїздками (відрядженнями) та інтелектуальні бізнес-застосунки. Службовці, які часто виїжджають у відрядження, витрачають величезну кількість часу й коштів на організацію своїх поїздок, їм доводиться орієнтуватися в часових поясах, що постійно змінюються, населених пунктах і розкладах. Мобільні програми можуть спростити їм роботу, надавши більше часу для основної роботи.

Крім того, мобільні інтелектуальні застосунки надають оперативні дані відповідальним особам. Із їхньою допомогою менеджери мають можливість відстежувати ключові показники діяльності та поточні бізнес-операції, а також використовувати сигнали тривоги, попередження та можливості складання звітів.

⚠ Особливе місце серед допоміжних бізнес-застосунків посідають мобільні застосунки управління подіями для Event-агентств. У роботі [8] подано методику розроблення мобільних застосунків для управління подіями, із метою використання Event-агентствами.

На рис. 1.3 показано фрагмент звичайної сторінки сайта на мобільному пристрої.

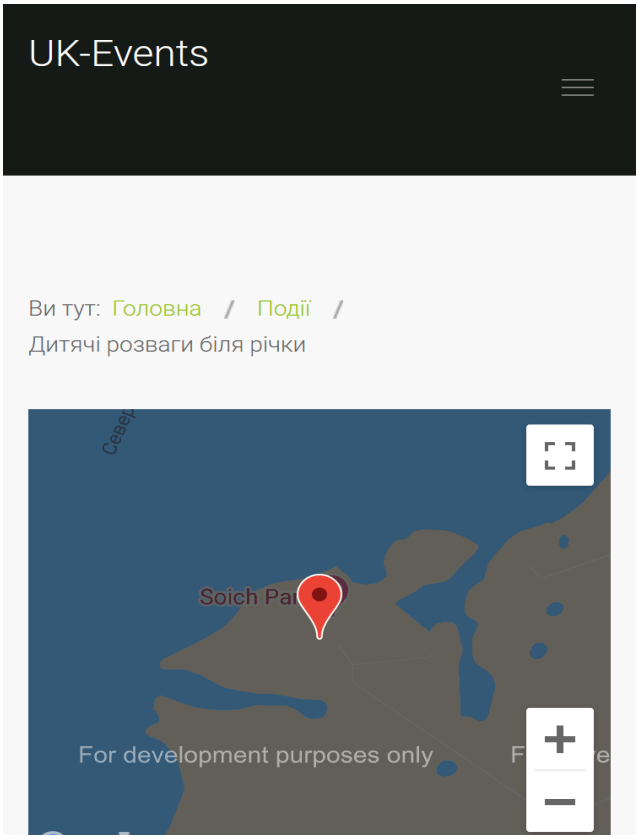


Рис. 1.3. **Фрагмент звичайної сторінки сайта на мобільному пристрої**

Для звичайної AMP-версії сторінки після проведення тестування було визначено такі результати:

- Загальна оцінка ступеня оптимізації сторінки для мобільних пристроїв – 98 зі 100.
- Поява першого контенту – 0,7 с.
- Індекс швидкості – 1,5 с.
- Поява найбільшого контенту – 2,3 с.
- Час до взаємодії – 2,0 с.
- Загальний час блокування – 70 мс.
- Зміщення макета – 0.

На рис. 1.4 показано фрагмент AMP-сторінки сайта на мобільному пристрої.

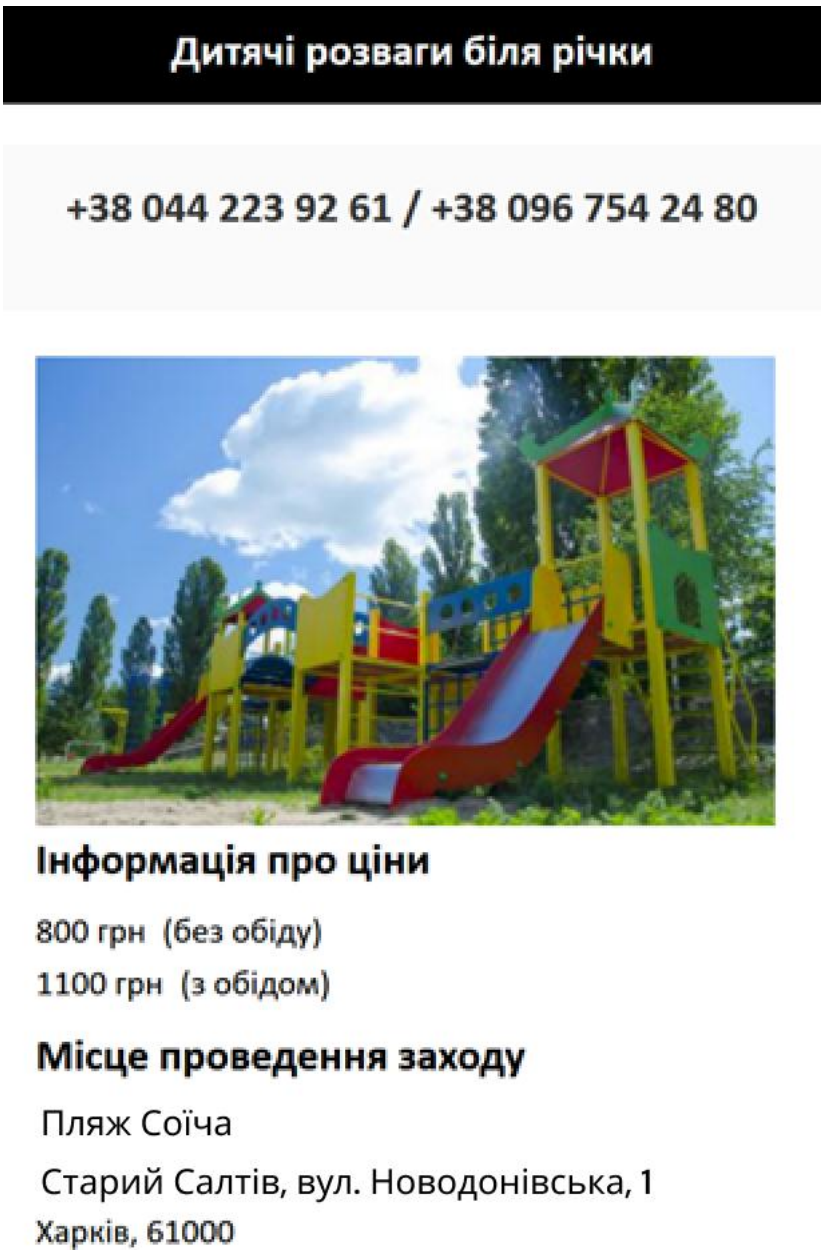


Рис. 1.4. **Фрагмент AMP-сторінки сайта на мобільному пристрої**

У результаті тестування для комп'ютерів пристроїв було визначено такі дані.

Для звичайної версії сторінки:

- Загальна оцінка ступеня оптимізації сторінки – 78 зі 100.
- Поява першого контенту – 1,5 с.
- Індекс швидкості – 1,8 с.
- Поява найбільшого контенту – 2,2 с.
- Час до взаємодії – 1,7 с.
- Загальний час блокування – 0 мс.
- Зміщення макета – 0,072.

Для AMP-версії сторінки:

- Загальна оцінка ступеня оптимізації сторінки – 100 зі 100.
- Поява першого контенту – 0,2 с.
- Індекс швидкості – 0,4 с.
- Поява найбільшого контенту – 0,6 с.
- Час до взаємодії – 0,6 с.
- Загальний час блокування – 10 мс.
- Зміщення макета – 0.

На підставі цього можна зробити висновок, що AMP-версія сторінки завантажується набагато швидше, ніж звичайна версія сторінки, що підкреслює доцільність використання мобільних застосунків.

Практична складова до підрозділу 1
Створення прототипу мобільного застосунку за допомогою онлайн-конструкторів

Мета – опанування студентами креативних умінь створення мобільного застосунку зі структурованою інформацією.

Завдання: Використовуючи онлайн-конструктор Appsfera.com, створити мобільний застосунок зі структурованою інформацією про себе та свій бізнес, яка б надавала максимально можливу ефективність подання та сприйняття інформації людиною. Для цього продумати структуру інформації, її оформлення й оформлення елементів інтерфейсу застосунку. Слід також ураховувати обмеження, які надає відповідний конструктор, та внутрішні алгоритми мобільної операційної системи.

Слід звернути увагу, що нині в конструктора **Appsfera.com** є безкоштовний спосіб перевірки програми в робота, але ця можливість може бути вимкненою або модифікованою. Не слід одразу намагатися оплатити передплату.

Процес створення програми

Після авторизації (Реєстрації) у сервісі, потрапляймо до свого особистого кабінету (https://my.appsfera.com/). Тут можна ознайомитися з документацією, прочитати FAQ та інші нюанси. Переходьмо за посиланням "Перейти в конструктор" (рис. 1.5).

Рис. 1.5. **Вікно переходу в конструктор**

Клацаймо на кнопку "Новий проєкт" і вводьмо його назву (рис. 1.6).

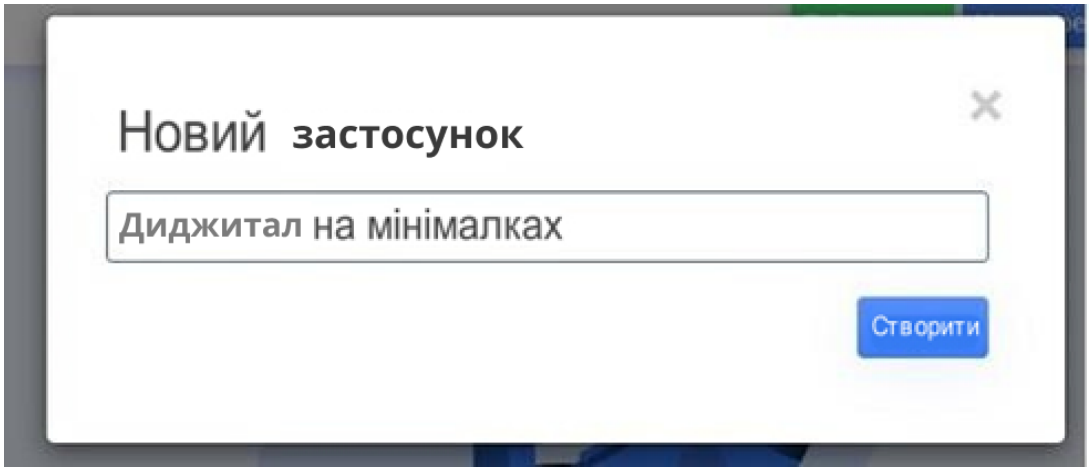


Рис. 1.6. **Створення нового проєкту**

Далі можна вибрати потрібний шаблон, з огляду на категорії конкретного виду бізнесу (рис. 1.7).

Рис. 1.7. **Вибір категорії конкретного виду бізнесу**

Далі відкривається вікно із чотирма основними моментами для створення програми: макет, дизайн, функції, налаштування (рис. 1.8).

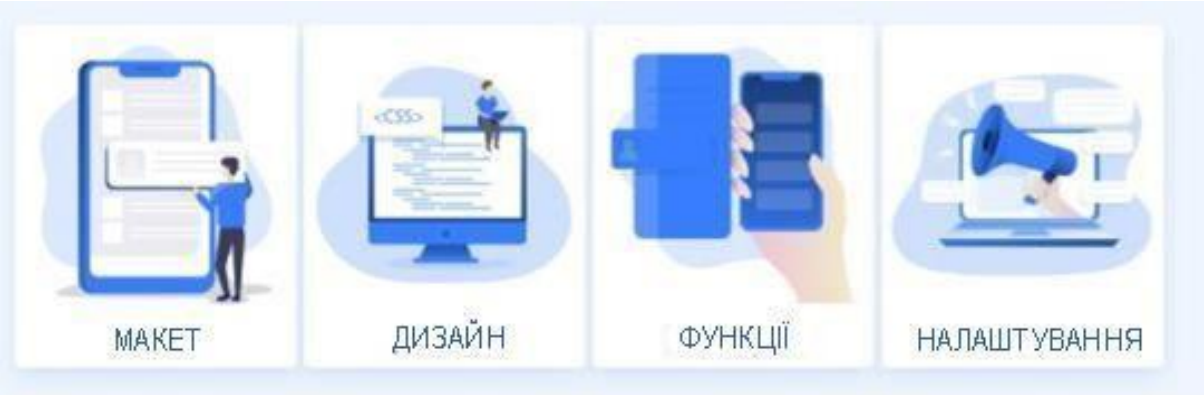


Рис. 1.8. **Вікно із чотирма основними моментами для створення програми**

Створення макета показано на рис. 1.9.

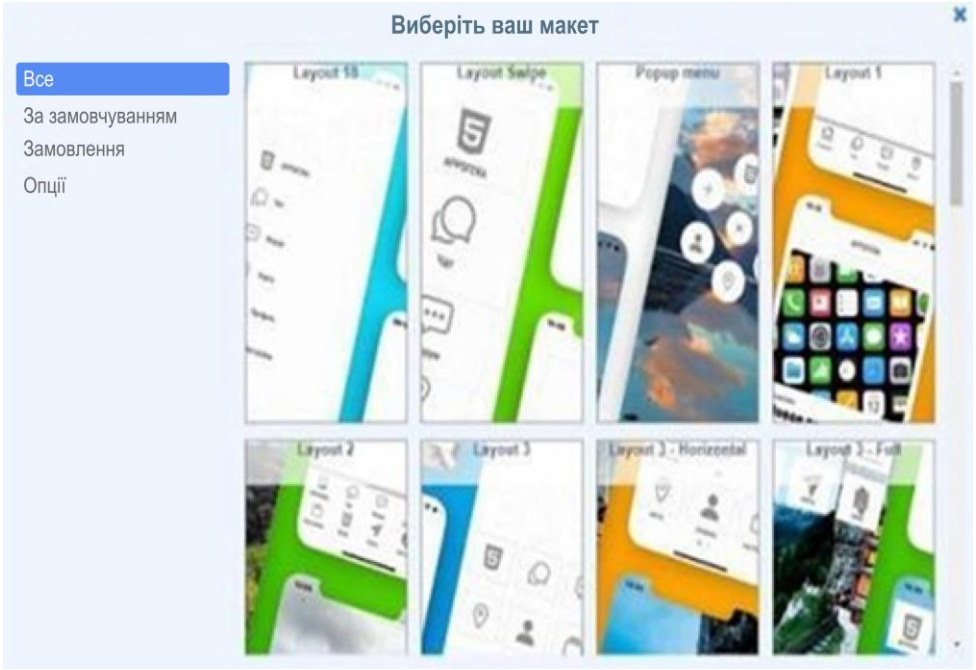


Рис. 1.9. **Створення макета**

Далі можна вибрати фон програми (рис. 1.10). Слід зауважити, що фон потрібно вибирати розміром не менше ніж 2 732 × 2 732 px, інакше він не змінюється. До того ж праворуч завжди буде зразкове

зображення того, що в результаті виходить, там можна клікати на рисунки. Це допомагає орієнтуватися в дизайні розроблення.

Рис. 1.10. **Вибір фону програми**

Потім переходьмо до категорії "Дизайн". Тут робімо налаштування вигляду UI-набору – кнопок, іконок, стилю вікон тощо (рис. 1.11).

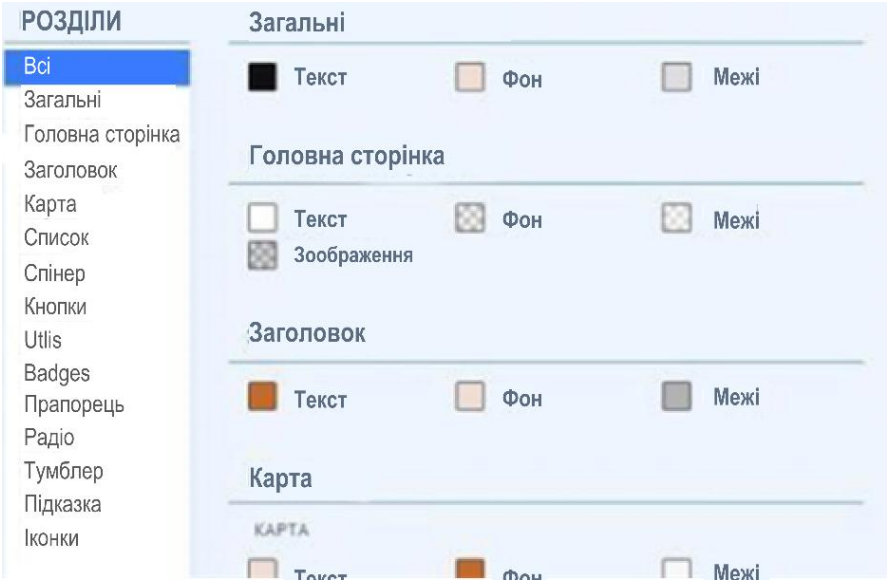


Рис. 1.11. **Налаштування вигляду UI-набору**

Дизайн самих іконок можна буде змінити далі (рис. 1.12).

Рис. 1.12. **Результати змінення дизайну іконок**

У категорії "Функції" (рис. 1.13) можна відредагувати/видалити вже наявні функції або додати нові. Саме тут іконку можна замінити або додати свою власну іконку.

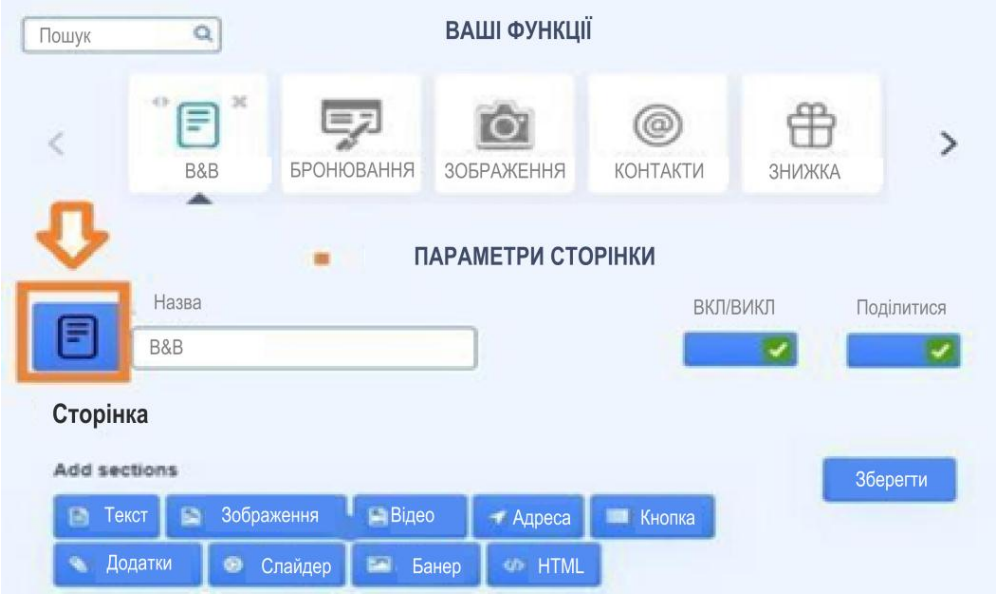


Рис. 1.13. **Категорії "Функції"**

Каталог доступних функцій показано на рис. 1.14.

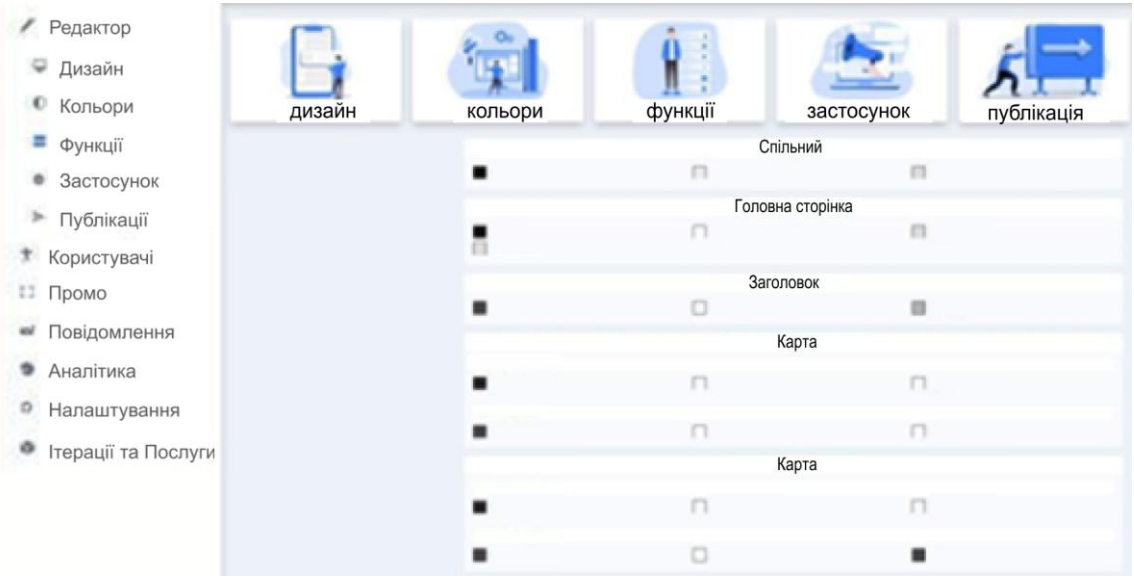


Рис. 1.14. **Каталог доступних функцій**

Загалом простір для творчості є величезним. У нашому прикладі застосунок вийшов лише з однією функцією (рис. 1.15).

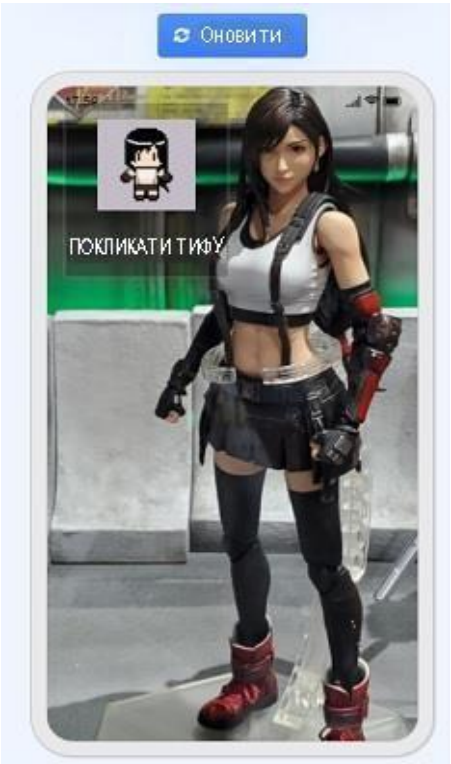


Рис. 1.15. **Результат реалізації однієї функції застосунку**

Наприкінці йдуть "Налаштування". Тут вибираймо іконку програми, яка буде радувати око користувача на його робочому столі (рис. 1.16).

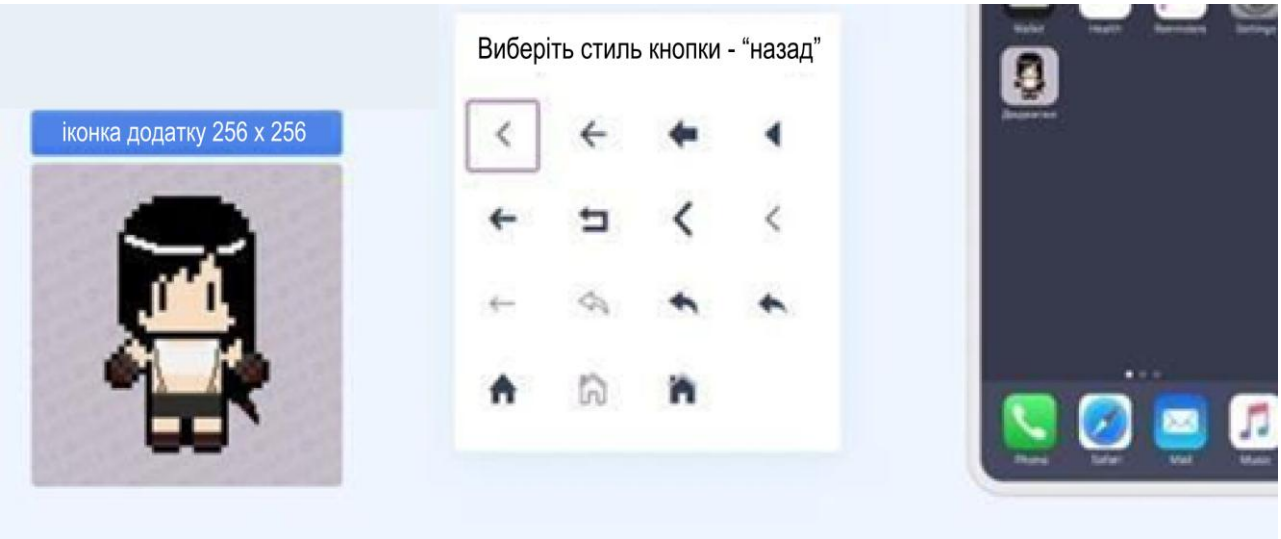


Рис. 1.16. **Вибір іконки програми**

Тепер фінал – клацаймо на зелену кнопку "Публікація" у правому верхньому кутку. "Публікація" – це лише надання вихідного коду програми або .apk-файл, тобто мова не йде про автоматичну публікацію в Google Play або AppStore.

Слід звернути увагу, нині в цього конструктора є безкоштовний спосіб перевірки програми в робота, але ця можливість може із часом бути вимкненою чи модифікованою. Публікація передбачає розміщення застосунку в офіційних магазинах Android та iOS, що з навчальною метою не потрібно. Не поспішайте оплачувати передплату. Нині протестувати програму можна за посиланням, зазначеним на рисунку. Це посилання можна відкрити на комп'ютері або, скопіювавши на смартфон, відкрити у браузері. Функціональність буде максимально наближеною до реального мобільного застосунку (рис. 1.17).

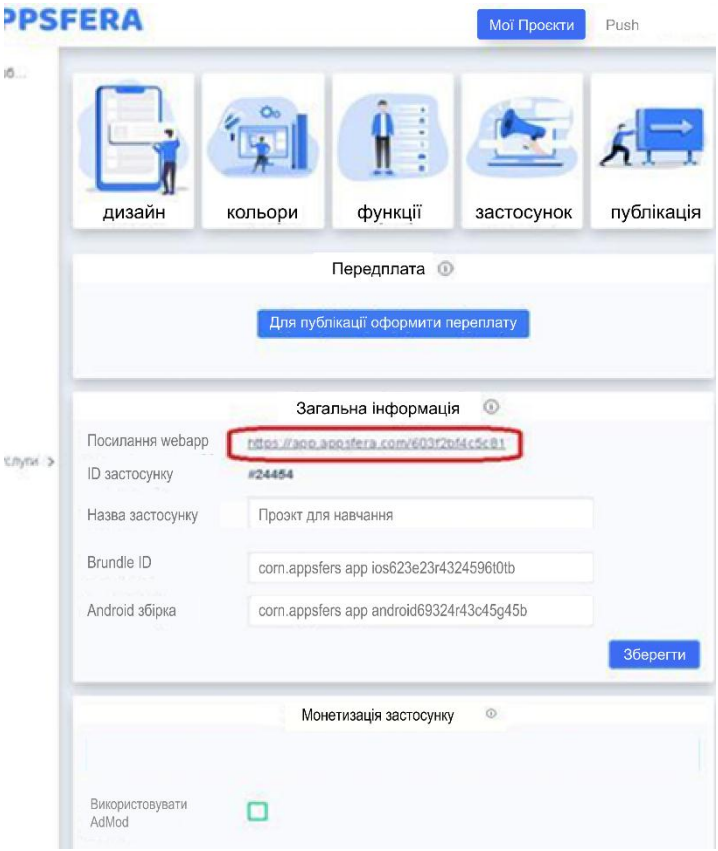


Рис. 1.17. **Публікація застосунку**

Мінімальними вимогами до мобільного застосунку є такі:

Кількість форм (функцій) у застосунку має бути не меншою ніж 5.

На кожній формі має бути не менше 5 рядків тексту чи 5 графічних елементів.

Дизайн та оформлення всіх елементів мають відрізнятися від закладеного за замовчуванням, включно з іконкою програми.

Інформацію, що надають у застосунку, має бути максимально пов'язаною з вами особисто або діяльністю, якою ви особисто займаєтеся.

Результати виконання лабораторної роботи надають у файлі Word чи PDF.

У звіті з лабораторної роботи слід подати такі моменти:

1) макет мобільного застосунку у внутрішньому поданні конструктора appsfera.com.

Усю інформацію подають у вигляді скріншотів після закінчення всіх робіт;

2) відомості із профілю https://app.appsfera.com/admin/account/. Приклад показано на рис. 1.18;

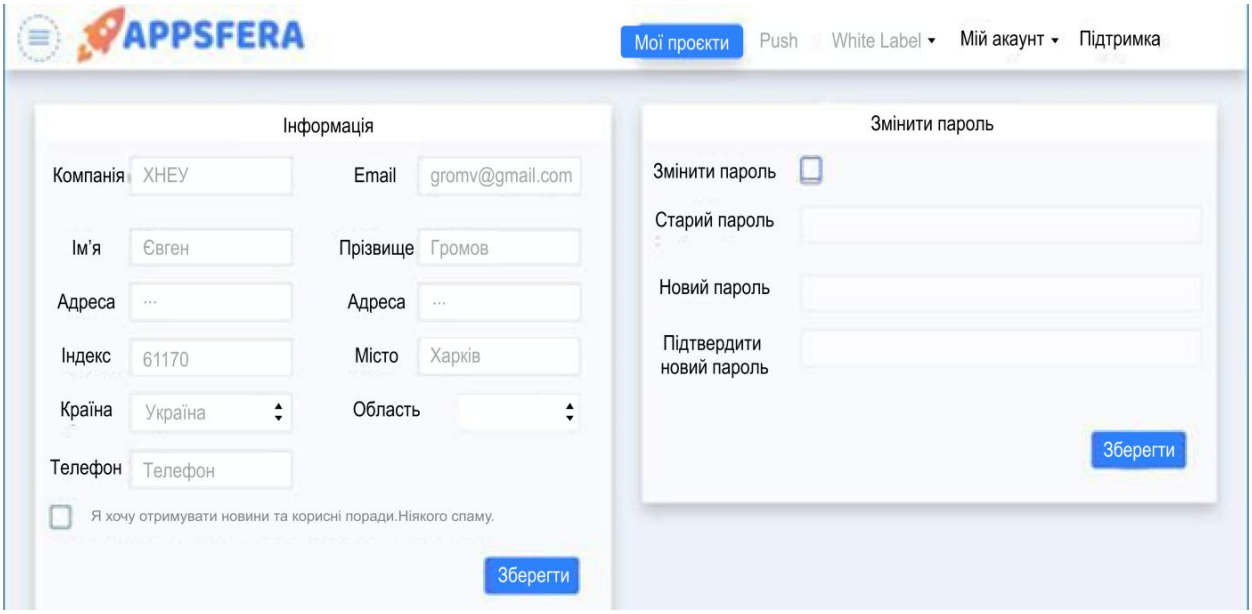


Рис. 1.18. **Приклад відомостей із профілю**

3) відомості з категорії "Дизайн". Приклад показано на рис. 1.19;

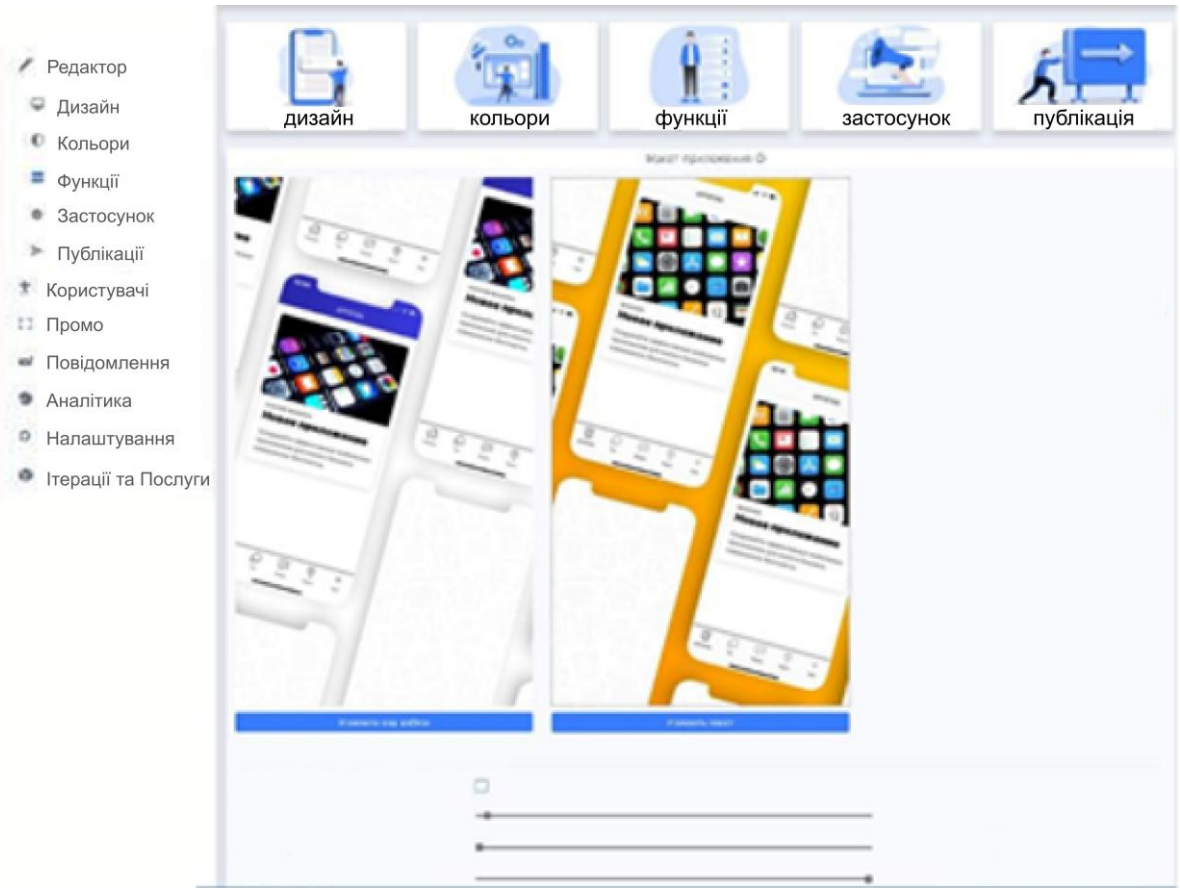


Рис. 1.19. **Приклад категорії "Дизайн"**

4) відомості з категорії "Кольори". Приклад показано на рис. 1.20;

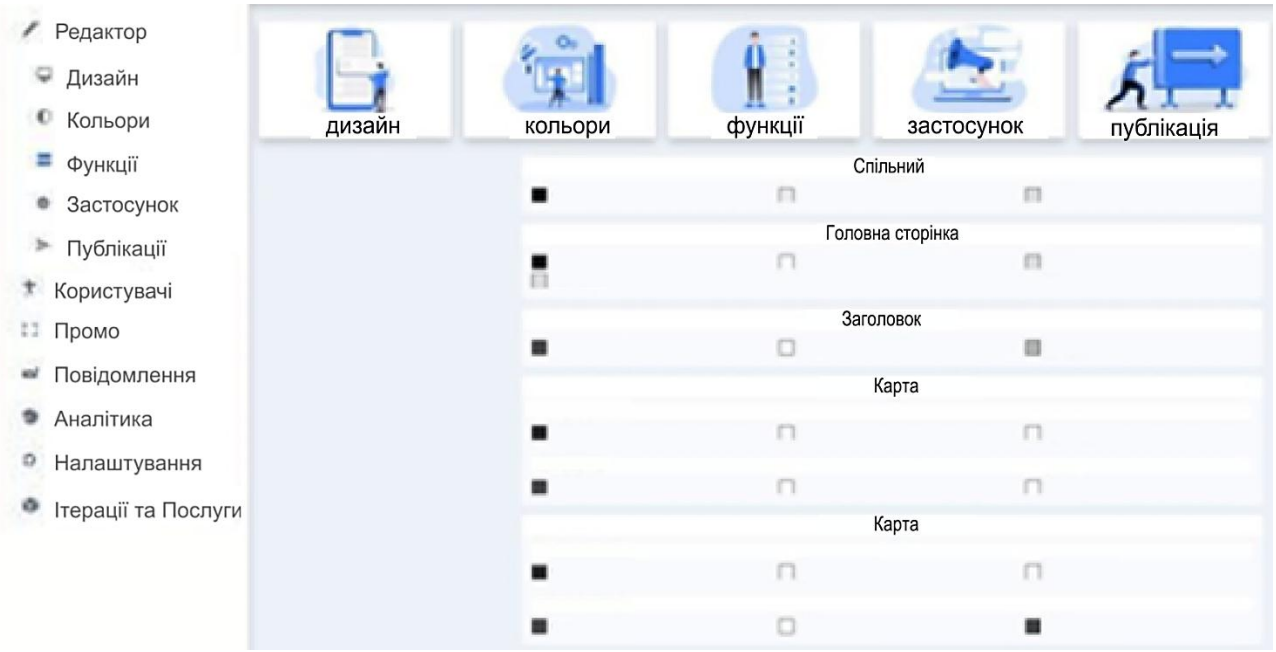


Рис. 1.20. **Приклад категорії "Кольори"**

5) відомості з категорії "Функції". Приклад показано на рис. 1.21;

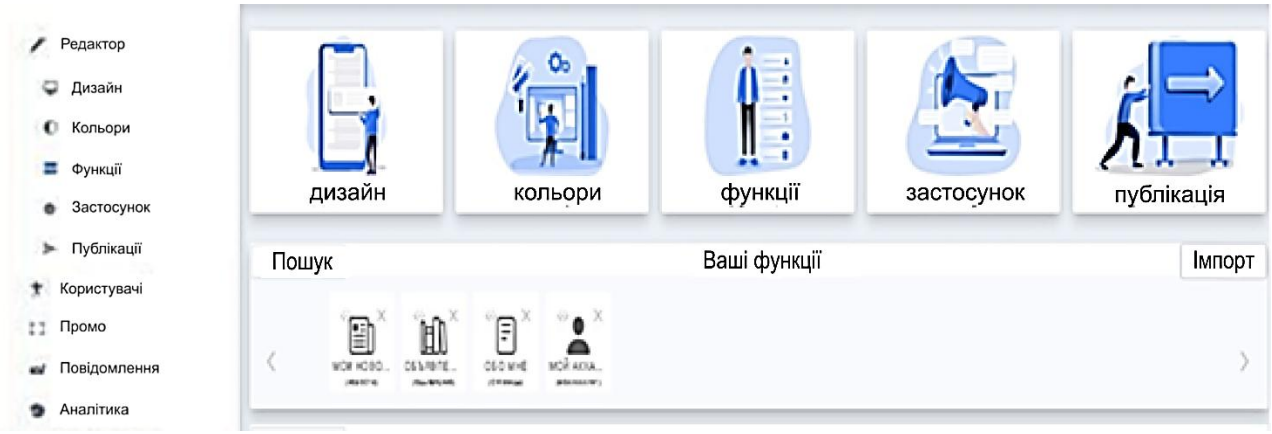


Рис. 1.21. **Приклад категорії "Функції"**

6) відомості з категорії "Застосунок". Приклади показано на рис. 1.22 і рис. 1.23.

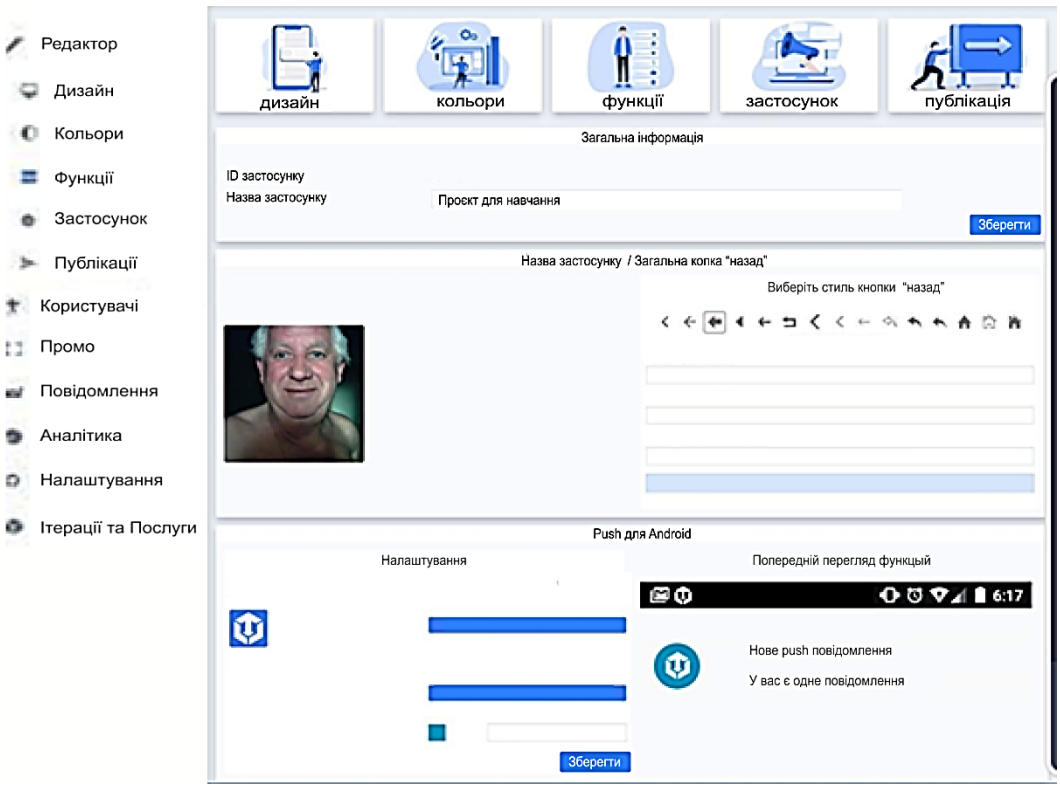


Рис. 1.22. **Приклад категорії "Застосунок"**

Рис. 1.23. **Приклад результату налаштування зміни зображення в категорії "Застосунок"**

7) результати тестування мобільного застосунку. Надати посилання на вебверсію застосунку, наприклад *https://app.appsfera.com/603f2bf4c5c81*.

Посилання подати у вигляді тексту для можливості перегляду. Усю інформацію подають у вигляді скриншотів після закінчення всіх робіт.

Контрольні запитання для самоперевірки

1. Проаналізуйте поняття та загальні особливості мобільних застосунків.
2. Назвіть сфери застосування мобільних застосунків.
3. У чому особливість функцій мобільних застосунків?

Рекомендована література: [1; 3; 9; 11; 13 – 17].

2. Архітектура мобільних застосунків

Мета – дослідження архітектурних особливостей мобільних застосунків.

Професійні компетентності: здатність аналізувати структуру та контент проєктів інтерактивних медіа; здатність оцінювати та забезпечувати якість виконуваних робіт.

2.1. Особливості реалізації архітектури "клієнт – сервер" для мобільних застосунків

⚠ Архітектуру програми часто створюють для ілюстрації загальних властивостей програмного забезпечення (наприклад, коду програми та платформи) й обладнання (наприклад, клієнта, сервера та мережевих пристроїв). У літературі виділяють низку базових шаблонів.

⚠ Архітектуру застосунків, зазвичай, проєктують у термінах архітектури "клієнт – сервер", коли один або кілька клієнтських пристроїв запитують інформацію із сервера. Сервер зазвичай відповідає необхідній інформації (рис. 2.1).

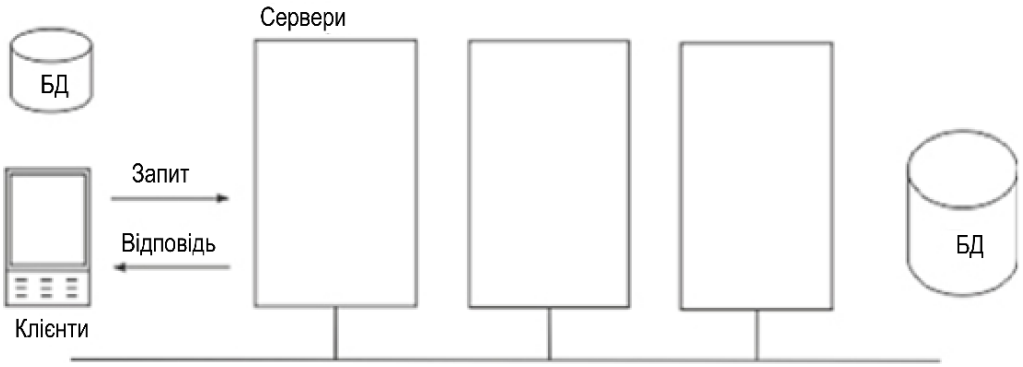


Рис. 2.1. **Клієнт-серверна архітектура**

Розгляньмо такі поняття клієнт-серверної архітектури, як: шари, рівні та зв'язки між шарами й рівнями.

Шари. Функціональність коду по всьому застосунку необов'язково є рівномірною. Деякі розділи коду програми краще підходять для опрацювання інтерфейсу користувача, тоді як інші розділи розроблено для управління бізнес-логікою або з'єднання з базою даних або серверних систем.

Розшарування визначає розподіл робіт усередині коду програми на одній машині. Найчастіше шари – це не більш ніж програмні модулі, розміщені в різних папках або каталогах на боці клієнта або сервера.

Із боку клієнта зазвичай є від нуля до трьох шарів у коді програми. Із боку сервера – від одного до трьох шарів коду застосунків.

Частково це важливо для якісного проєктування програмного забезпечення, яке забезпечує повторне використання коду, частково для безпеки, а частково із міркувань зручності.

Клієнт із нульовою кількістю шарів коду, насправді, не має спеціального коду програми. Цей тип клієнта зазвичай згадують як **тонкого клієнта** і він є можливим в архітектурі "клієнт – сервер", за якого сервер містить весь код користувача програми. Клієнта з одним до трьох шарів коду програми зазвичай називають **товстим клієнтом**.

Сервер також може містити від одного до трьох шарів спеціального коду програми. Однак за визначенням не може бути нульової кількості шарів коду на сервері.

Шар, код якого найбільш тісно взаємодіє з користувачем, часто згадують як шар подання. Другий шар часто називають рівнем бізнес-логіки, оскільки він зазвичай опрацьовує бізнес-логіку коду. Третій шар

часто називають рівнем доступу до даних. Він зазвичай опрацьовує зв'язок із базою даних або джерелом даних (рис. 2.2).

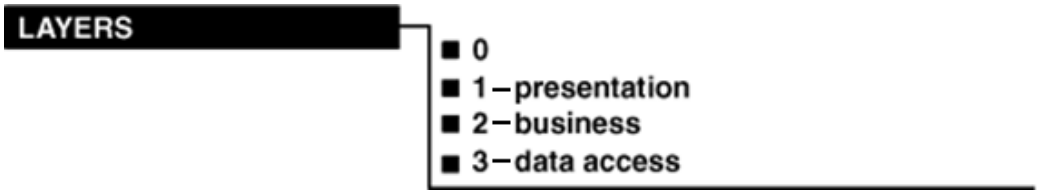


Рис. 2.2. **Шари**

Є цілком можливою наявність більш ніж трьох рівнів на боці клієнта або сервера, але занадто багато шарів може призвести до труднощів під час управління. У результаті це зустрічають не часто.

Рівні. Розподіл функціональності коду програми на шари допомагає повторному використанню, але це не робить архітектуру автоматично масштабованою. Для того щоб це зробити, важливо розподіляти код на декількох машинах.

Рівень описує розподіл робіт прикладного коду на кількох машинах. Багаторівневість, зазвичай, передбачає розміщення програмних модулів на різних машинах у розподіленому серверному середовищі. Якщо код програми вже розподілено за шарами, це робить багаторівневість набагато більш простим процесом.

Код, який найтісніше взаємодіє з користувачем, часто розташовано на рівні подання. Другий рівень, який містить логіку бізнес-застосунків та доступу до даних, часто згадують як рівень застосунків. Третій рівень зазвичай містить базу даних або джерело даних і його згадують як рівень бази даних (рис. 2.3). Це приклад трирівневої архітектури.

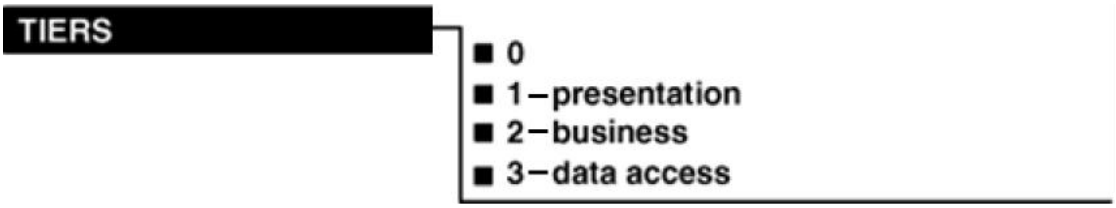


Рис. 2.3. **Рівні**

Сервери, що входять до кожного рівня, можуть відрізнятися як за можливостями, так і за номерами. Наприклад, у великомасштабному розподіленому середовищі вебзастосунку може бути велика кількість вебсерверів на рівні подання, менша кількість серверів застосунків на прикладному рівні та два кластеризовані сервери баз даних на рівні баз даних. Можливість додавання додаткових серверів часто називають *горизонтальним масштабуванням*. Можливість додавання більш потужних серверів часто називають *вертикальним масштабуванням*. Багаторівневість прикладного коду в таких випадках значно полегшує можливості масштабування застосунків.

У великомасштабних розподілених вебзастосунках рівні часто обмежено брандмауерами. Наприклад, брандмауер може бути розміщено перед рівнем подання, тоді як другий брандмауер – перед рівнем застосунків. Отже, рівень подання міститься між брандмауерами в так званій *демілітаризованій зоні* (ДЗ), тоді як рівні програми та бази даних сервера захищено другим брандмауером і розташовано в так званій зоні *інтрамережі*. Отже, масштабування сприяє безпеці та дозволяє великим підприємствам захистити важливі внутрішні системи від трафіка, що виходить із ненадійних зон, як-от інтернет та ДЗ. Без масштабованості дуже важко захистити внутрішні системи.

Клієнт. Зазвичай, мобільні пристрої можуть працювати як тонкі або товсті клієнти або їх можуть бути розроблено таким способом, що вони можуть бути гостами вебсторінок (рис. 2.4).

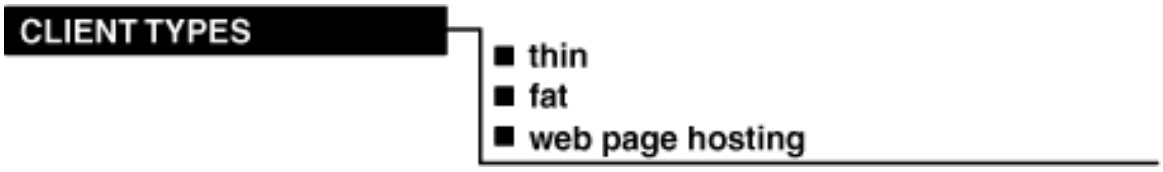


Рис. 2.4. **Типи клієнтів**

Тонкий клієнт. Тонкі клієнти не мають прикладного коду користувача та їхня функціональність повністю забезпечено сервером (рис. 2.5). Отже, вони не залежать від операційної системи мобільного або мобільного пристрою типу товстого клієнта.

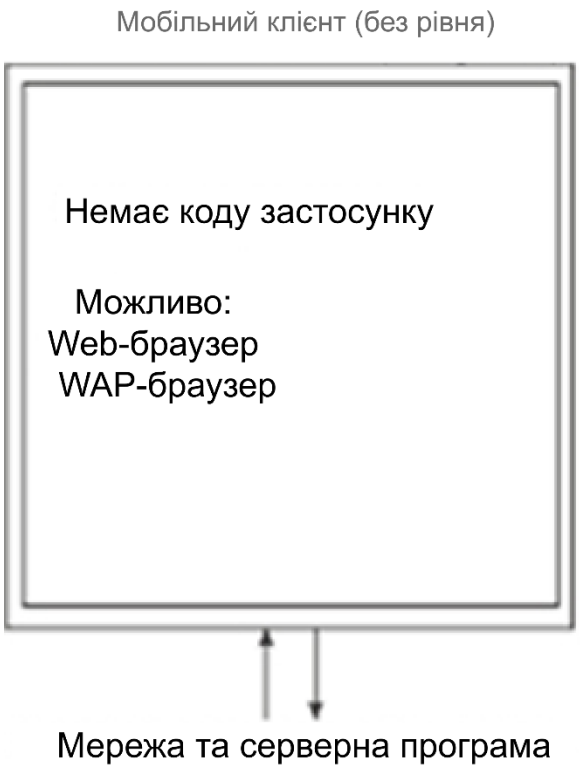


Рис. 2.5. **Тонкий клієнт**

Тонкі клієнти зазвичай використовують веббраузери та протокол бездротового доступу (WAP), щоб відображати такі типи сторінок прикладного змісту:

• Web (наприклад, HTML, XML);

• WAP (наприклад, WML).

Наприклад, для відображення вебсторінок КПК може відображати їх через Microsoft Pocket Internet Explorer, а планшетні та портативні ПК також можуть відображати їх через Microsoft Internet Explorer або Google Chrome. Аналогічно WAP-браузер на мобільному телефоні може відображати WML-сторінки.

Тонкі клієнти мають низку переваг перед товстими клієнтами. Наприклад, вони є набагато простішими в обслуговуванні та підтриманні, оскільки не мають прикладного коду та даних. У результаті немає необхідності розглядати версії прикладного коду та механізми їхнього розподілу клієнтові.

Проте труднощами під час роботи з тонкими клієнтами є те, що вони мають бути на постійному зв'язку із сервером, оскільки це їхнє джерело для оновлення та здобуття даних. Якщо зв'язок не є надійним, можливо,

замість цього типу клієнта потрібно буде розглянути автономні товсті клієнтські застосунки.

Товстий клієнт. Товсті клієнти зазвичай мають від одного до трьох шарів прикладного коду й можуть працювати незалежно від сервера протягом певного періоду часу.

Зазвичай, товсті клієнти є найбільш корисними в ситуаціях, якщо зв'язок між клієнтом та сервером не може бути гарантованим. Наприклад, товста клієнтська програма може бути в змозі приймати введення користувача та зберігати дані в локальній базі даних, поки відновлює зв'язок із сервером і дані може бути переміщено на сервер. Це дозволяє користувачеві продовжувати працювати, навіть якщо він/вона перебуває поза контактом із сервером.

Однак товсті клієнти значною мірою залежать від операційної системи та типу мобільного пристрою, тому можуть бути труднощі за повторного використання та поширення коду. Ви також можете мати кілька версій коду для підтримання кількох пристроїв.

Товсті клієнти можуть бути реалізованими за допомогою одного, двох чи трьох шарів прикладного коду. Отже, загалом краще використовувати два або, переважно, три шари, щоб використовувати якомога більшу частину прикладного коду (рис. 2.6).

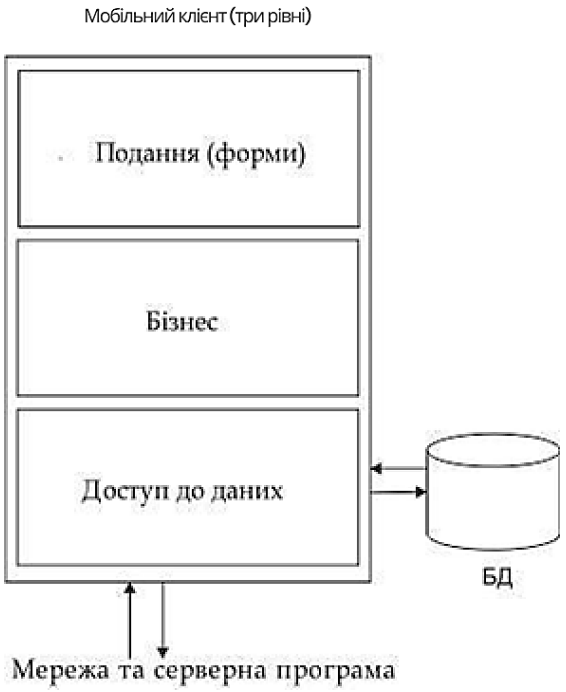


Рис. 2.6. **Товстий клієнт – три шари**

Якщо використовувати тільки один шар, украй важко виділити окремі сфери/області функціональності та повторно використовувати та поширювати код на кілька типів пристроїв.

2.2. Гостинґ вебсторінок

Крім того, на мобільному пристрої можна відображати та опрацьовувати вебсторінки, навіть якщо мобільний клієнт лише періодично під'єднаний до мережі та серверних систем. Для того щоб це зробити, на мобільному пристрої є потрібним еквівалент "міні" вебсервера.

Наприклад, компанія Microsoft випустила HTTP-сервер, який працює на мобільному клієнті якраз із такою метою. Дані, уведені користувачем на весторінці, опрацьовують HTTP-сервером і зберігають в локальній базі даних, якщо не можуть бути завантаженими на сервер, доки з ним не відновлять з'єднання.

Клієнти, які використовують гостинґ вебсторінки, також можуть мати від одного до трьох шарів (рис. 2.7). Головна різниця між вебсторінками товстого клієнта та вікнами Windows полягає в тому, що рівень подання відображає та використовує вебсторінки, а не форми Windows.

Рис. 2.7. **Гостинґ вебсторінки – три шари**

Сервер. Архітектура сервера, зазвичай, складається з одного-трьох шарів коду, реалізованого на одному-трьох рівнях. Хоча є намір завжди будувати трирівневу архітектуру, вона має плюси та мінуси. Наприклад, велика трирівнева архітектура може бути досить дорогою для реалізації. Якщо програму орієнтовано на обмежену кількість користувачів, трирівнева архітектура може бути надмірною.

Однорівнева архітектура. Однорівнева архітектура передбачає розміщення всіх трьох шарів на одному сервері (рис. 2.8). Є кілька переваг і недоліків такого типу архітектури. *Переваги*: дуже зручна, швидке розроблення та розгортання. *Недоліки*: невисока масштабованість і безпека.

З одного боку, дуже зручно розробляти код на одній машині, з іншого – дуже важко масштабувати програму. Для вебпрограми також важко захистити сервер за допомогою брендмауерів і зон безпеки, оскільки сервер майже напевно буде у віддаленій зоні, що може призвести до високого ризику безпеки доступу до бази даних.

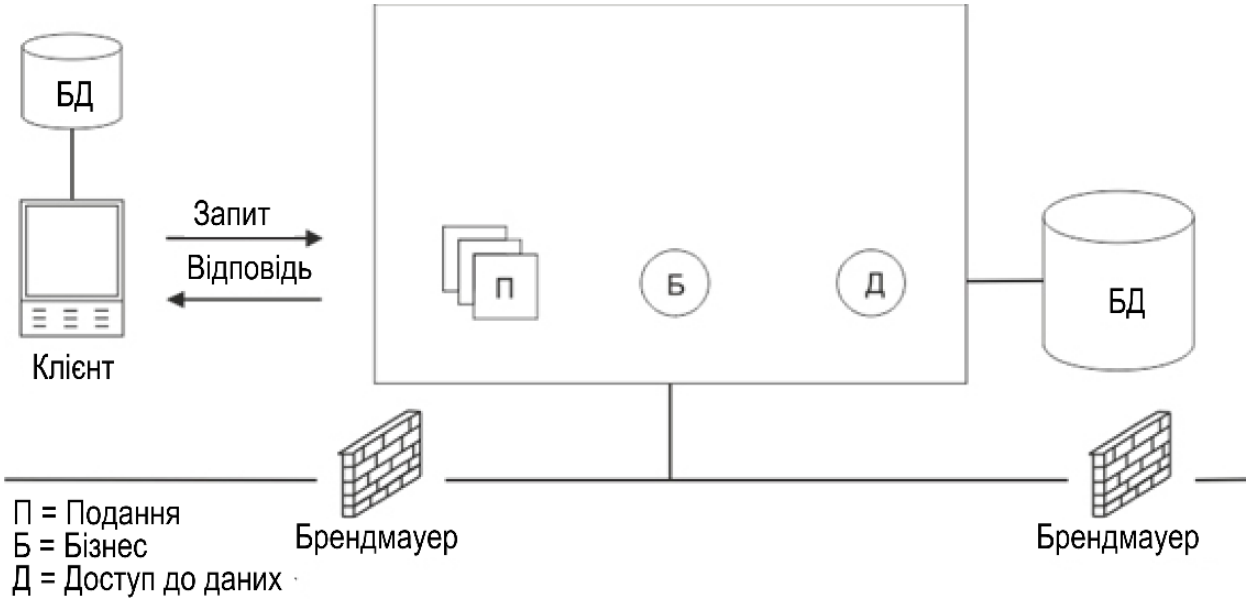


Рис. 2.8. **Однорівнева архітектура**

Дворівнева архітектура. Під час реалізації дворівневої архітектури сервер бази даних відокремлюють від сервера застосунків (рис. 2.9). Є кілька плюсів і мінусів такого підходу.

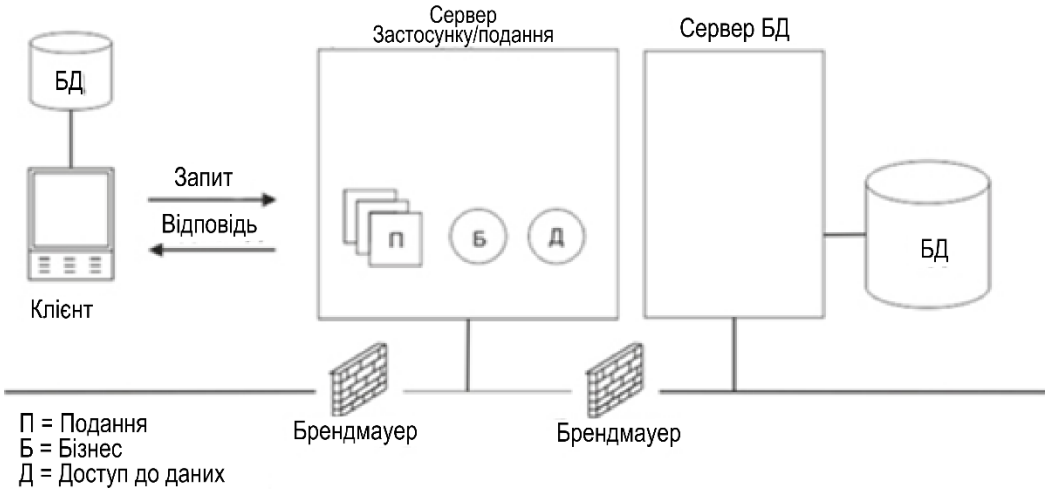


Рис. 2.9. **Дворівнева архітектура**

Плюси: зручність і можливість спеціалізації сервера баз даних.

Мінуси: нижча масштабованість, труднощі в разі забезпечення безпеки та дорожнеча.

Відокремлення сервера баз даних дозволяє йому стати більш спеціалізованим, але, як і раніше, украй складно застосовувати масштабування. За такої архітектури важко захистити сервери із брендмауерами та зони безпеки, хоча цей розподіл відбувається краще, ніж в одноріневій архітектурі. Проте безпека програми все одно є невисокою.

Трирівнева архітектура. Трирівнева архітектура забезпечує розподіл сервера баз даних, сервера застосунків та сервера подання один від одного (рис. 2.10). Такий підхід має свої переваги та недоліки.

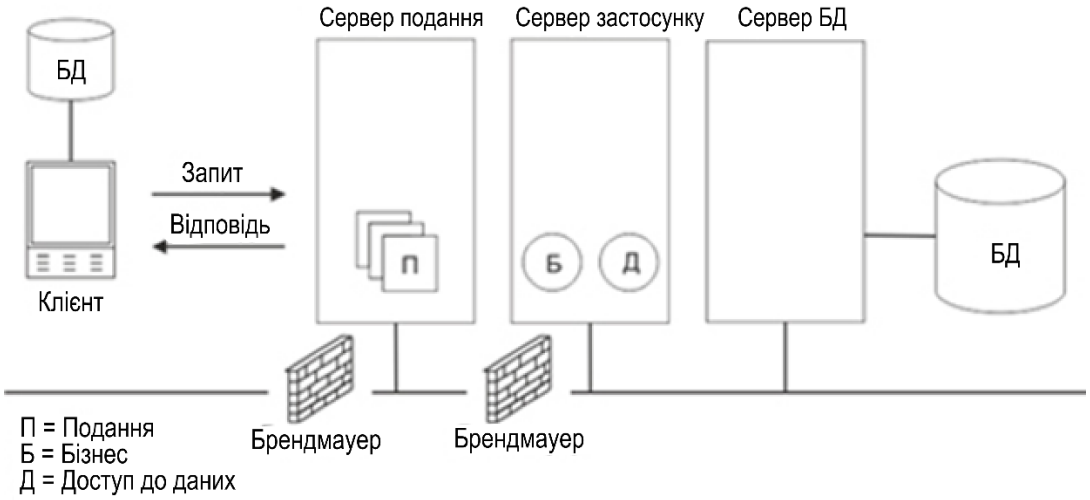


Рис. 2.10. **Трирівнева архітектура**

Переваги: масштабованість, захист за допомогою брендмауерів і зон, можливість спеціалізації сервера баз даних.

Недоліки: надмірність, складніше розвиватися, важче управляти, висока дорожнеча.

Відокремлення бази даних дозволяє серверу бази даних стати більш спеціалізованим сервером. Відокремлення презентаційного та прикладного серверів також дозволяє виконати спеціалізацію цих серверів, що в сукупності дає величезний потенціал масштабованості. Також можна краще захистити програму з використанням брендмауерів та зон ДЗ.

2.3. Типи з'єднань

Мобільні пристрої зазвичай працюють в одному із трьох режимів: завжди на зв'язку, є частково пов'язаними та ніколи не під'єднаними (рис. 2.11). Ці режими докладніше описано далі.

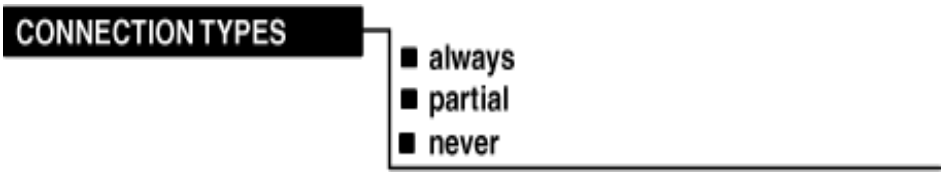


Рис. 2.11. **Способи під'єднання**

Постійне під'єднання. Мобільні пристрої, як-от смартфони або айфони, зазвичай, працюють у режимі постійного зв'язку. Мобільні пристрої, насправді, стали розширенням наявних програм та інфраструктури, що дозволяють користувачам завжди бути з'єднаними із програмами під час роботи.

Часткове під'єднання. Іноді мобільні пристрої можуть бути вимкненими протягом тривалого часу. Наприклад, співробітники мобільних офісів можуть періодично під'єднуватися до сервера в офісі для отримання електронної пошти, контактної інформації або завдань, які потрібно зробити. Працівник вимикає мобільний пристрій та виконує роботу поза офісом, під час якого він може посилатися на скачану інформацію. Користувач також може локально оновлювати інформацію на своєму мобільному пристрої перед повторною синхронізацією мобільного пристрою із сервером пізніше.

Без під'єднання. Наявні також мобільні пристрої, які ніколи не під'єднано до серверів, наприклад, ігрові пристрої.

Синхронізація

⚠ Тип з'єднання впливає на те, як можна синхронізувати дані між мобільним пристроєм та серверами. Синхронізацію реалізують двома основними способами: безперервно або методом із проміжним зберіганням.

Коли між клієнтом і сервером безперервне з'єднання, синхронізація даних між клієнтом та сервером також є *безперервною* й може бути досягнута синхронним або асинхронним способами (рис. 2.12).

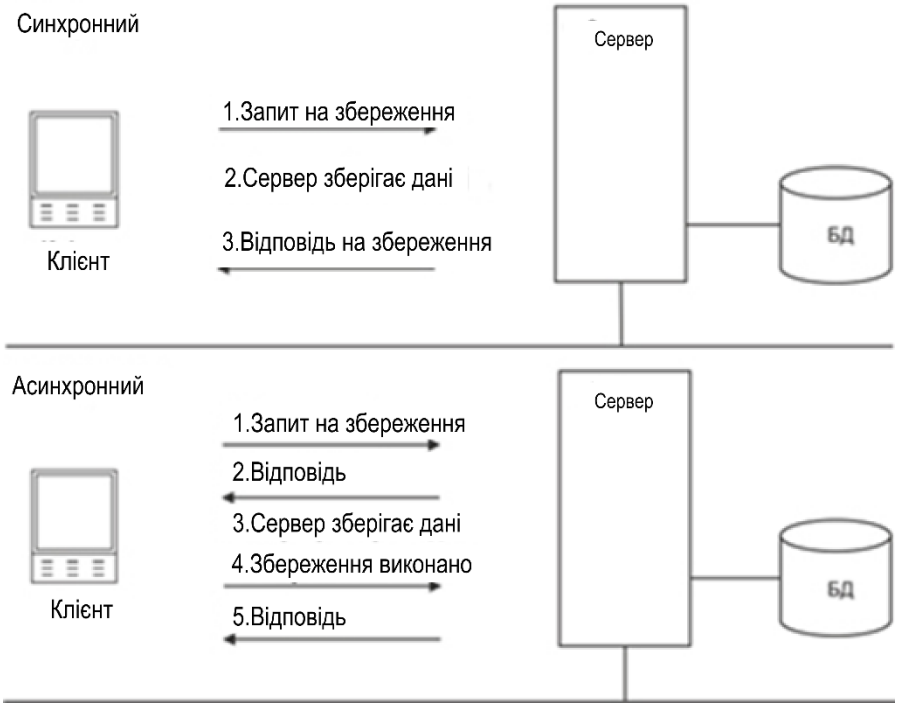


Рис. 2.12. **Синхронна та асинхронна взаємодія**

⚠ *Синхронне з'єднання* виникає, коли запит для зберігання даних надсилають на сервер, на якому зберігають дані. Потім дані розміщують у сховищі, як-от бази даних на сервері. За синхронного з'єднання всі дані зберігають повністю перед тим, як сервер підтверджує отримання даних і звільняє інтерфейс користувача клієнта.

⚠ *Асинхронне з'єднання* відбувається, коли спочатку на сервер надсилають запит для зберігання даних, а потім дані буде збережено.

Дані розміщують у сховищі, як-от база даних на сервері. Однак за асинхронного зв'язку дані не може бути повністю збережено до розпізнавання сервером клієнта. Зазвичай сервер спочатку розпізнає запит клієнта й лише потім зберігає дані. Після виконання запиту на зберігання сервер повідомить про це клієнта.

Метод із проміжним зберіганням

⚠ Коли не може бути гарантовано з'єднання між клієнтом і сервером, є можливість безпечно зберігати та передавати інформацію, використовуючи метод із проміжним зберіганням. Припустімо, що мобільний користувач бажає ввести дані тоді, коли його мобільний пристрій не під'єднано до сервера. Програма мобільного клієнта може спочатку зберегти дані в локальній базі даних. Пізніше, коли з'єднання з'явиться, мобільний застосунок передасть дані з локальної бази даних базі даних на сервері (рис. 2.13).

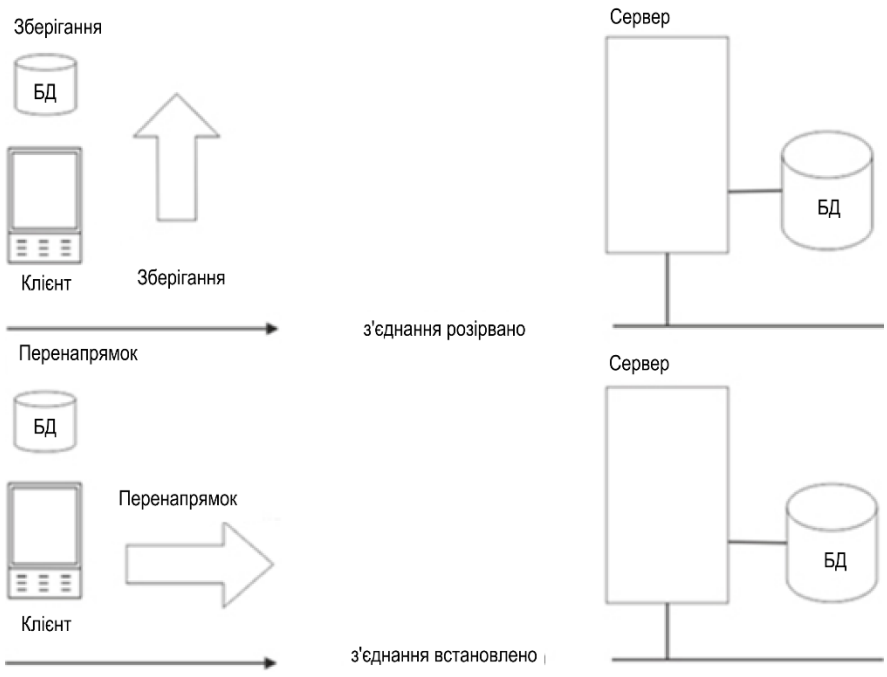


Рис. 2.13. **Метод із проміжним зберіганням**

Важливо, що якщо адміністратор дозволить мобільним користувачам таким способом зберігати дані в локальній базі даних, необхідно також забезпечити цілісність даних, коли синхронізують дані із сервером

бази даних, оскільки інші користувачі можуть додати або змінити суперечливі дані на мобільних пристроях.

Архітектурні шаблони. Якщо припустити, що є чотири можливі шари клієнта, три рівні сервера та три типи під'єднання, то, загалом, виходить 36 можливих комбінацій. Однак не всі із цих комбінацій є корисними. Нині найпоширеніша модель – це "часткове під'єднання", оскільки зв'язок не завжди може бути гарантованим.

Нульова трирівнева архітектура безперервного під'єднання

Якщо мобільний клієнт не має шарів прикладного коду, це означає, що він є тонким клієнтом. Сервер зберігає весь код програми та має трирівневу архітектуру. На рис. 2.14 показано просту мобільну архітектуру.

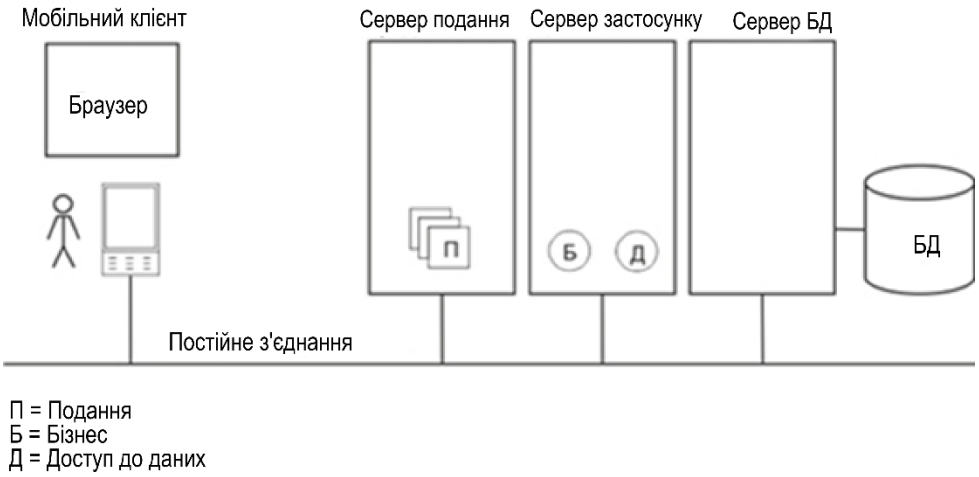


Рис. 2.14. **Нульова трирівнева архітектура безперервного під'єднання**

Рівень подання має прикладний код, який може відображати сторінки мобільного пристрою. Звичайні вебсторінки (наприклад, ASP.NET, JSP, HTML) є доступними для перегляду за допомогою веббраузера, наприклад Microsoft Pocket Internet Explorer.

Рівень подання також взаємодіє з даними й має доступ до об'єктів на рівнях програми та бази даних. Зазвичай, дані може бути прочитано з бази даних і назад записано під час оновлення.

Ця архітектура є дуже простою, тому що передбачено, що мобільний клієнт завжди буде під'єднаним до сервера. Отже, немає необхідності зберігати дані програми на мобільному пристрої. Якщо мобільний

пристрій буде вимкнено, то не буде можливості здобути всю актуальну інформацію, доки з'єднання не буде відновлено.

Три шари, три рівні, архітектура часткового під'єднання

Мобільний клієнт має три шари коду прикладної програми, тобто він є товстим клієнтом. Сервер також містить код застосунку та його організовано у вигляді трирівневої архітектури. На рис. 2.15 показано більш складну мобільну архітектуру.

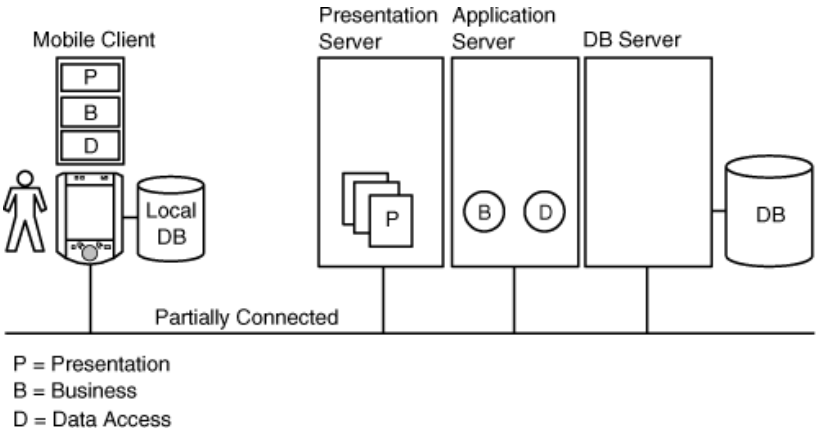


Рис. 2.15. **Складна мобільна архітектура**

Мобільний клієнт має повністю автономний застосунок, який може читати та записувати дані, що вводить користувач, до локальної бази даних, коли він не є під'єднаним до сервера. Коли відновлюють зв'язок, дані може бути отримано з локальної бази даних і завантажено на сервер за допомогою механізму проміжного зберігання даних.

2.4. Принципи розроблення гарної архітектури

Для успішного розроблення мобільних застосунків слід дотримуватися певних принципів створення гарної архітектури. Насправді, може бути неможливим досягнення всіх цих принципів. Однак багато кращих архітектур мобільних застосунків задовольняють більшість із цих принципів.

Метод незалежності. В ідеальному варіанті мобільні програми має бути розроблено незалежними від типу пристрою та від платформи,

наскільки це можливо. Це не завжди легко чи можливо, але гарні застосунки, зазвичай, написано так, що вони можуть працювати на багатьох пристроях та платформах.

Насправді, більшість застосунків не відповідають цим парадигмам. Імовірно, вам буде необхідно вибрати найбільш відповідний пристрій і платформу та, відповідно, писати програми. Отже, вам, майже напевно, доведеться вибирати мобільні пристрої, як-от смартфони чи айфони. Кожен пристрій і платформа мають різні характеристики, що необхідно брати до уваги під час розроблення застосунків.

Висока продуктивність та доступність. Архітектура мусить, зазвичай, мати відмінну продуктивність за нормального та пікового періодів потреби в ресурсах. Наприклад, для сайта брокерського майданчика електронного бізнесу в день пікової торгівлі акціями. Якщо люди можуть використовувати сайт у будь-який момент часу, архітектура мусить мати також високу доступність.

Масштабованість. Архітектура має бути масштабованою, щоб її швидко адаптувати за можливого значного збільшення кількості користувачів, застосунків та функціональних можливостей. Архітектуру має бути розроблено так, щоб легко дозволити горизонтальне (додаткові сервери) та вертикальне (додавання більш швидких серверів) масштабування без шкоди для будь-яких наявних застосунків.

Системні вимоги користувача. Архітектура, зазвичай, має дозволяти опрацьовувати якомога більші типи та більшу кількість користувачів. Наприклад, вебзастосунки з великим обсягом графіки для відображення на смартфонах можуть бути змістовними, але якщо користувачі мають лише низькошвидкісні лінії з'єднання, продуктивність не буде задовільною. Отже, слід мати на увазі повний спектр користувачів для високо- та низькопродуктивних систем.

Практична складова до підрозділу 2
Проєктування архітектури багатошарового застосування

Мета – вивчення структури багатошарового застосування, освоєння процесу моделювання предметної галузі для проєктування архітектури прикладних мобільних **застосунків** та набуття навичок у розробленні прототипу архітектури.

Теоретичні засади роботи

Опис основних елементів архітектури (завд. 1)

Створення архітектури мобільних застосунків є процесом формування структурованого рішення, що відповідає всім технічним та операційним вимогам і забезпечує оптимальні загальні атрибути якості: як-от продуктивність, безпека та керованість.

Цей процес містить ухвалення низки рішень на підставі широкого діапазону чинників (ці рішення буде ухвалювати сам студент, вирішуючи конкретні завдання, зазначені у практичній частині роботи). Кожне із цих рішень може мати істотний вплив на якість та продуктивність мобільних застосунків, а також зручність їхнього використання.

Як і будь-яка інша складна структура, мобільні застосунки мають будуватися на міцному фундаменті. Помилки із проєктуванням ключових сценаріїв, неправильне розроблення загальних питань або нездатність спрогнозувати довгострокові наслідки ключових рішень можуть поставити під загрозу всю програмну систему мобільних застосунків. Сучасні інструменти та платформи спрощують завдання щодо створення застосунків, але не усувають потреби в ретельному їхньому проєктуванні на підставі конкретних сценаріїв і вимог.

Проєктування систем мають здійснювати з урахуванням потреб користувача, системи (IT-інфраструктури) та бізнес-цілей. Для кожної із цих складових визначають ключові сценарії та виділяють важливі параметри якості (наприклад, надійність або масштабованість), а також основні сфери задоволеності та незадоволеності. Якщо можна, необхідно виробити врахувати показники успішності в кожній із цих сфер. У практичній частині роботи цьому присвячено завд. 1. Основна рекомендація під час вирішення завд. 1 – визначати компроміси та знаходити баланс між конкурентними вимогами цих трьох сфер. Наприклад, інтерфейс рішення користувача дуже часто визначено бізнес-цілями та IT-інфраструктурою загалом, і зміни одного із цих компонентів можуть істотно впливати на результівні механізми взаємодії з користувачем. Аналогічно, зміни у вимогах щодо взаємодії з користувачем можуть сильно впливати на вимоги до бізнес-сфери та IT-інфраструктури.

Отже, архітектор має враховувати загальний ефект від ухвалених проєктних рішень, обов'язково наявні компроміси між атрибутами якості та компроміси, необхідні для виконання користувальних, системних і бізнес-вимог для мобільних застосунків.

Проєктування прототипу архітектури (завд. 2)

Під час проєктування архітектури мобільних застосунків слід ураховувати, що архітектура має об'єднувати бізнес- та технічні вимоги через сценарії – варіанти використання (Use Case) із подальшим визначенням можливих реалізацій в алгоритмах.

Прототип архітектури має ґрунтуватися на вимогах (визначених із попереднього пункту), які впливають на структуру застосунку.

Слід ураховувати ознаки якісної архітектури, яка знижує бізнес-ризики, пов'язані зі створенням технічного рішення, і має значну гнучкість, щоб справлятися із природним розвитком технологій як в обладнанні та програмному забезпеченні, так і для користувачів сценаріїв і вимог.

Під час проєктування архітектури слід керуватися такими основними принципами [2]:

• Створювати з готовністю до змін – слід продумати, як із часом може знадобитися змінити програму, щоб вона відповідала вимогам і завданням, що виникають, і передбачити необхідну гнучкість.

• Створювати моделі для аналізу та скорочення ризиків – використовувати засоби проєктування, системи моделювання, як-от уніфікована мова моделювання (Unified Modeling Language, UML), і засоби візуалізації, якщо необхідно виявити вимоги, ухвалити архітектурні та проєктні рішення і проаналізувати їхні наслідки. Однак не створювати надто формалізовану модель, вона може обмежити можливості для виконання ітерацій та адаптації дизайну.

• Використовувати моделі та візуалізації як засоби спілкування під час спільної роботи. Використовувати моделі, подання та інші способи візуалізації архітектури для ефективного обміну інформацією та зв'язку з усіма зацікавленими сторонами, а також для забезпечення швидкого сповіщення про зміни дизайну.

• Розглянути можливість використання інкрементного й ітеративного підходу під час роботи над архітектурою. Слід починати з базової архітектури, відтворюючи повну картину, а потім треба опрацювати можливі варіанти в ході ітеративного тестування та доопрацювання архітектури.

Отже, під час створення прототипу (завд. 2) необхідно пам'ятати, що архітектура має:

розкривати структуру системи, але приховувати деталі реалізації;

реалізовувати всі варіанти використання та сценарії;

якщо можна, відповідати всім вимогам різних зацікавлених сторін; виконувати вимоги як за функціональністю, так і за якістю.

Застосування компонентів на всіх шарах програми, що розробляють (завд. 3)

Архітектуру мобільних застосунків можна описати як організацію чи структуру системи, де система становить набір компонентів, що виконують певну функцію чи набір функцій. У цьому сенсі основне завдання архітектури полягає у створенні компонентів, із метою забезпечення певної функціональності. Таку організацію функціональності часто називають групуванням компонентів за функціональними сферами [2].

На рис. 2.16 показано типову архітектуру мобільних застосунків, компоненти якого згруповано за функціональними сферами, але слід ураховувати, що функціональні сфери використовують не тільки для групування компонентів, а можуть бути застосовними для реалізації взаємодії та організації спільної роботи компонентів.

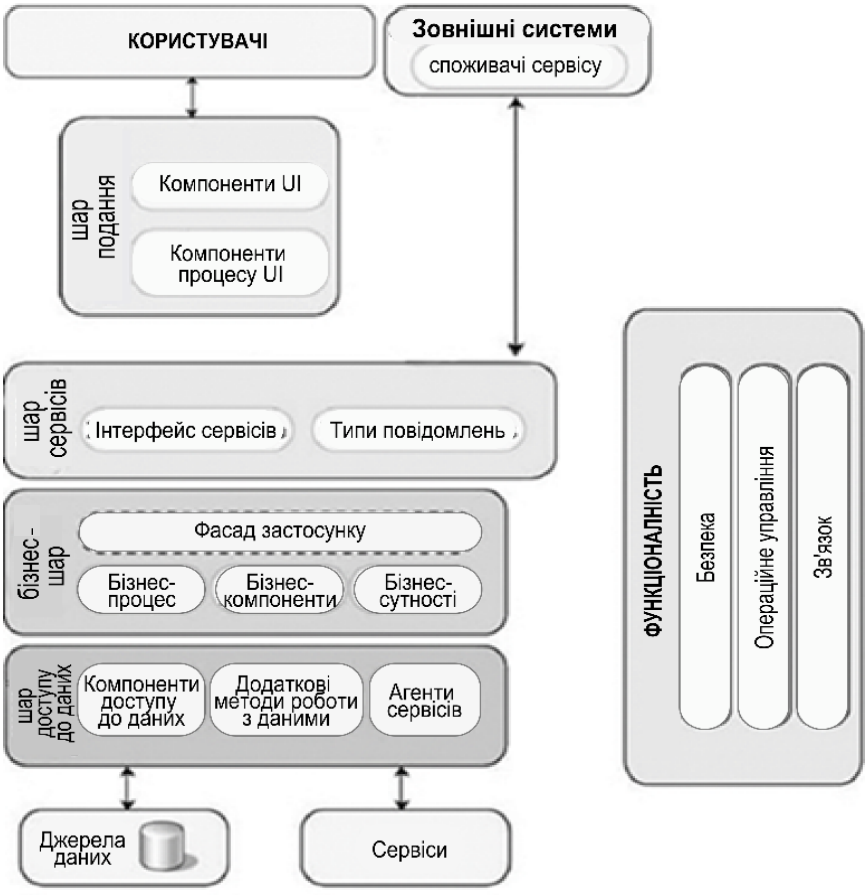


Рис. 2.16. **Типова архітектура програми**

Під час вирішення завд. 3 (проєктування архітектури мобільних застосунків як взаємодії компонентів) слід ураховувати рекомендації щодо проєктування різних функціональних сфер [2]:

• Розподіляйте функції. Треба розподілити програмну частину мобільних застосунків на окремі компоненти з урахуванням мінімального перекриття функціональності. Важливим чинником є граничне зменшення точок дотику, що забезпечить високу згуртованість (high cohesion) і слабкий зв'язок (low coupling). Неправильне розмежування функціональності може призвести до високого зв'язку та труднощів взаємодії, навіть незважаючи на слабке перекриття функціональності окремих компонентів. Основна перевага такого підходу – незалежна оптимізація функціональних можливостей і ще, перебій однієї з функцій не призведе до перебою інших, оскільки їх можна виконувати, незалежно одна від одної. Такий підхід також спрощує розуміння та проєктування мобільних застосунків та полегшує управління складними взаємопов'язаними системами.

• Використовуйте принцип відповідальності. Кожен окремий компонент або модуль має відповідати тільки за одну конкретну властивість/функцію або сукупність пов'язаних функцій.

• Використовуйте принцип мінімального знання (Law of Demeter, LoD). Компоненту чи об'єкту не мають бути відомими внутрішні деталі інших компонентів чи об'єктів.

• Не повторюйтеся (Don't repeat yourself, DRY). Намір має бути позначено лише один раз. У застосуванні до проєктування програми це означає, що певну функціональність має бути реалізовано лише в одному компоненті й не мають дублювати в жодному іншому компоненті.

• Мінімізуйте проєктування наперед. Проєктуйте лише те, що потрібно. Якщо вимоги до застосування чітко не визначено, або є ймовірність зміни дизайну із часом, намагайтеся не витрачати багато сил на проєктування завчасно. Цей принцип називають YAGNI (You ain't gonna need it).

• Визначте чіткий контракт компонентів. Компоненти, модулі та функції мають визначати контракт або специфікацію інтерфейсу, що чітко обумовлює їхнє використання та поведінку. Контракт має описувати, як інші компоненти можуть виконувати доступ до внутрішньої функціональності компонента, модуля або функції й поведінку цієї функціональності з погляду попередніх умов, постумов, побічних ефектів, винятків, робочих характеристик та інших чинників.

• Максимально ізолюйте наскрізну функціональність від бізнес-логіки програми. Наскрізна функціональність містить аспекти безпеки, обміну інформацією або керованості, як-от протоколювання й інструментування. Змішування коду, що реалізує ці функції, із бізнес-логікою може призвести до створення дизайну, який буде складно розширювати й обслуговувати. Унесення змін до наскрізної функціональності буде потребувати перероблення всього коду бізнес-логіки.

Отже, важлива мета архітектора під час проєктування мобільних застосунків полягає в максимальному спрощенні дизайну через його розподіл на функціональні сфери. Наприклад, до різних функціональних сфер можна зарахувати інтерфейс користувача (user interface, UI), виконання бізнес-процесів і доступ до даних. Компоненти в кожній із цих сфер мають реалізовувати конкретну функціональність і не змішувати код різних функціональних сфер. Так, у компонентах UI не має бути коду прямого доступу до джерела даних – для здобуття даних у них мають використовувати або бізнес-компоненти, або компоненти доступу до даних.

Практичні завдання

Завдання 1. Сформуйте опис основних елементів архітектури (користувача, системи, бізнес-завдання) і для кожного елементу сформулюйте ключові сценарії та виберіть показники якості.

Дайте розгорнуті відповіді на такі запитання:

1. Як користувач буде використовувати мобільний **застосунок**?

2. Як мобільний застосунок будуть розгортати та обслуговувати під час експлуатації?

3. Які вимоги до атрибутів якості, як-от безпека, продуктивність, можливість паралельного опрацювання, інтернаціоналізація та конфігурація, ставлять до мобільного застосунку?

4. Як спроєктувати застосунок, щоб він залишався гнучким і зручним в обслуговуванні протягом тривалого часу?

5. Які є основні архітектурні напрями, які можуть впливати на застосунок зараз чи після його розгортання?

Завдання 2. Розробіть прототип архітектури мобільного застосунку. Запропонуйте варіант розподілу застосунку на шари.

Під час проєктування архітектури необхідно відповісти на такі запитання:

1. Які частини архітектури є фундаментальними, зміна яких у разі неправильної реалізації становить найбільші ризики?

2. Які частини архітектури, найімовірніше, зазнають змін, і навіть проєктування яких елементів можна відкласти?

3. Наведіть основні припущення щодо архітектури програмного комплексу.

4. Які умови можуть призвести до реструктуризації дизайну?

5. Чи є рішення про залежність між шарами та потоками даних між ними?

Завдання 3. Розробіть попередні пропозиції щодо застосування компонентів на всіх шарах мобільного застосунку, який розробляють. Перелічіть, які функції ці компоненти мають виконувати, із метою досягнення необхідної функціональності.

Під час проєктування компонентів архітектури мобільного застосунку необхідно відповісти на такі запитання:

1. Які важливі (на цьому етапі проєктування) функції (властивості) системи можна доручити окремим компонентам?

2. Чи можна зараз зарахувати компоненти до певного типу?

3. Як об'єднати окремі компоненти з функціональних сфер?

4. Чи дублюють функціональність у різних компонентах?

5. Які індивідуальні експлуатаційні вимоги до компонентів можна назвати на цьому етапі?

6. Чи має кожен метод, викликаний компонентом, достатні відомості про те, як опрацьовувати запити, що надходять, а в разі необхідності, як переспрямовувати їх до відповідних підкомпонентів або інших компонентів?

7. Які передбачають сценарії розгортання програм мобільного застосунку, чи всі компоненти будуть виконувати в межах одного процесу, чи необхідно забезпечити підтримання зв'язку між компонентами, що розгортають у різних процесах?

8. Як компоненти будуть реалізовувати наскрізну функціональність?

Контрольні запитання для самоперевірки

1. Що таке "архітектура мобільного застосунку"?
2. Перелічіть особливості архітектури програмного забезпечення.
3. Які запитання має поставити архітектор під час розроблення структури мобільного застосунку?
4. Що таке "наскрізна функціональність"? Укажіть потенційні проблеми від змішування коду, що реалізує ці функції, із бізнес-логікою.

Рекомендована література: [2; 4; 5; 7; 12; 13; 15 – 18].

3. Особливості візуалізації інформації для використання в мультимедійних застосунках

Мета – дослідження технології візуалізації інформації для використання в мультимедійних застосунках.

Професійні компетентності: здатність застосовувати сучасні методи й інструменти для досліджень у сфері видавництва та поліграфії; здатність аналізувати структуру та контент проєктів інтерактивних медіа.

3.1. Аналіз основних завдань візуалізації інформації для використання в мультимедійних застосунках

Із розвитком сучасного суспільства почалася комп'ютерна ера – ера інформації та інформатизації. Щорічно в локальних і глобальних мережах з'являються сотні терабайтів інформації. Упроваджують різні механізми для пошуку потрібної інформації, проте ці засоби є ефективними тоді, коли користувачі мають конкретну мету й розуміють, яку інформацію як зберігають. В інших випадках допомогти користувачеві потенційно можуть методи візуалізації інформації.

Те, що візуалізація є потужним інструментом швидкого оцінювання ситуації, обґрунтовано двома важливими чинниками. По-перше, більшу частину інформації про навколишній світ здобуваємо саме за допомогою зору, і це природний процес сприйняття, який не потребує від людини жодних зусиль. По-друге, візуальні методи дозволяють чітко відобразити основний зміст, закладений у даних, незалежно від їхнього обсягу та структури. У результаті замість стомлювального процесу вивчення багатосторінкових звітів і таблиць, можна повністю сфокусуватися на розробленні подальшого плану дій.

⚠ Основним завданням візуалізації є використання зорового сприйняття людиною інформації для посилення її пізнавальних здібностей. Зорова система людини є здатною швидко опрацьовувати візуальні сигнали, а передові інформаційні технології перетворили комп'ютер на потужний засіб управління цифровою інформацією. Візуалізація є мостом, що зв'язує зорову систему людини та інформаційну систему, допомагаючи ідентифікувати образи, будувати гіпотези та вилучати ідеї з величезних масивів даних, що сприяє науковому дослідженню і прогнозуванню.

Для вирішення більшості бізнес-завдань, пов'язаних із дослідженням і візуалізацією масивів бізнес-даних, часто достатньо базових діаграм та їхніх підвидів [6]. Це обумовлено тим, що бізнес стикається з дуже обмеженим типом висновків, які можна зробити на кількісних даних: більше/менше, є/немає залежності, позитивна/негативна динаміка, максимальна/мінімальна частина тощо. А для цього основних типів графічних методів цілком достатньо.

Крім того, дані візуальні інструменти є абсолютно простими, наочними та зрозумілими будь-якій аудиторії. Вони не потребують спеціальних знань як для їхнього прочитання, так і для побудови; підтримують всіма наявними програмними продуктами, отже, можуть бути безперешкодно використаними в будь-якій ситуації.

З іншого боку, відчуття простоти та зрозумілості провокує використовувати такі діаграми, не замислюючись про специфіку даних, які буде подано аудиторії. А це велика помилка, оскільки окремі види діаграм не є вільно замінними та їх мають підбирати, із кінцевою метою аналізу.

Найбільш ефективним є застосування візуалізації інформації в різноманітних мультимедійних застосунках, що полегшить сприйняття великої кількості текстової інформації та буде сприяти її наочному поданню в системах мультимедіа.

3.2. Технологія візуалізації інформації для використання в мультимедійних застосунках

⚠ Перший етап технології візуалізації інформації для використання в мультимедійних застосунках має складатися із двох кроків:

1) формування переліку ключових термінів предметної галузі;

2) формування таблиць даних.

Крок 1.1. Формування переліку ключових термінів.

Сирі дані може бути подано в будь-якому вигляді, починаючи з електронних таблиць і закінчуючи неструктурованими текстами. Сирими даними, які буде використано у процесі візуалізації видавничих стандартів, будуть ключові терміни.

Крок 1.2. Формування таблиць даних.

 Другий етап технології візуалізації інформації для використання в мультимедійних застосунках складається з таких кроків:

1) візуальне відображення ключових термінів у вигляді піктограм;

2) візуальне відображення ключових термінів у вигляді зображень;

3) візуальне відображення таблиць даних.

Візуальні структури – це зображення, які повною мірою передають зміст ключового терміна. Для візуалізації інформації, із метою її подальшого використання в мультимедійних застосунках, необхідно застосовувати три типи візуальних структур:

1) перший тип – *піктограми*, які схематично зображують зміст ключового терміна;

2) другий тип – *зображення*, яке є прикладом використання ключових термінів на практиці;

3) третій тип – *інтерактивна інфографіка* (parameter ruler), за допомогою якої подано табличні дані.

Важливо вибрати найбільш виразну візуальну структуру, на якій відображено всі дані без утрат, яка легко та швидко може бути інтерпретованою людиною з найменшою кількістю помилок і передає по максимуму всі відмінності в даних. Візуальні структури першого типу розробляють, відповідно до наявних прикладів використання ключових термінів мультимедійного видавництва.

Другий тип візуальних структур теж розробляють власноруч, але деякі структури вже було розроблено та їх використовують у настільних мультимедійних видавничих системах.

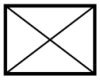
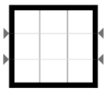
Третій тип візуальних структур становить інтерактивну інфографіку (parameter ruler), яка належить до одного з методів візуалізації, а саме metaphor visualization, за допомогою якої буде візуалізовано дані.

Програми електронного видавництва QuarkXPress, Adobe InDesign, Adobe PageMaker мають уже розроблені візуальні структури щодо ключових термінів у галузі додрукарської підготовки видань, зокрема щодо

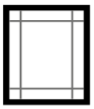
загальних понять, оформлення та виконання сторінки видання та поліграфічного виконання видання. Під візуальними структурами мають на увазі піктограми, які використовують як елементи інтерфейсу настільних видавничих систем, – це кнопки, які передають зміст ключового терміна в галузі мультимедійного видавництва. Візуальні структури, які передають зміст ключових термінів у галузі додрукарської підготовки видань, зокрема використовують не лише в настільних видавничих систем, а також у звичайному текстовому редакторі, як-от Microsoft Word. Набір піктограм формують на основі піктограм, поширених у настільних видавничих системах та текстових процесорах (табл. 3.1).

Таблиця 3.1

Аналіз наявних візуальних структур у настільних видавничих системах та текстових процесорах

Ключові терміни	Настільні видавничі системи та текстові процесори			
	Adobe PageMaker	Adobe InDesign	QuarkXPress	Microsoft Word
1	2	3	4	5
Сторінка видання				
Група шрифтів				
Абзацний відступ				Немає
Ініціал	Немає			
Ілюстрація				
Таблиця				
Текст				

Закінчення табл. 3.1

1	2	3	4	5
Накреслення вічка	N**B***I*	Немає	**B***I*	**Ж** ***К*** <u>**Ч**</u>
Поля	Немає	Немає	Немає	
Формат видання	Немає	Немає	Немає	

Здійснений аналіз уже наявних візуальних структур допоможе розробити всі інші структури, які не мають аналогів у настільних видавничих системах. Деякі візуальні структури, які було виявлено у процесі аналізу, або буде використано без змін, або до них буде внесено поправки. Відібрані та розроблені піктограми мають різний стиль відображення, деякі є плоскими, а деякі об'ємними, щоб подати їх у єдиному стилі було вибрано "плоский" дизайн (flat design).

"Плоский" дизайн (англ. **flat design**) – це мінімалістичний підхід до дизайну об'єктів, який підкреслює зручність використання, його більшою мірою орієнтовано на кінцевого користувача.

Дизайн цього мультимедійного додатка було розроблено згідно з новими тенденціями в галузі дизайну – "плоский" дизайн (flat design), про що говорять кольори й елементи навігації, використані в мультимедійному додатку.

Основні принципи flat design такі:

1) немає зайвих ефектів;

2) простота елементів;

3) акуратна робота зі шрифтами. Використання простих елементів підвищує важливість типографіки в дизайні. Роботу зі шрифтами мають виконувати дуже акуратно. Характер шрифту має доповнювати, а не суперечити загальній схемі дизайну. Шрифт у "плоскому" дизайні є важливим елементом навігації. Простота елементів не означає, що не можна використовувати складні шрифти. Просто все має бути витримано у стилі мінімалізму;

4) мінімалізм. У "плоскому" дизайні слід уникати зайвих "наворотів", складних підходів до візуалізації елементів.

У результаті застосування "плоского" дизайну щодо піктограм було визначено такі результати (рис. 3.1).

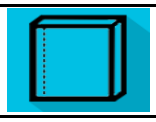
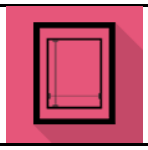
Блок видання	Видання	Початкова сторінка	Текст
			
Складний зошит	Зошит	Формат сторінки видання без полів	Група шрифтів
			
Складена обкладинка	Підстава	Фронтиспис	Титул
			

Рис. 3.1. **Піктограми у стилі flat design**

⚠ Третій етап технології інформації для використання в мультимедійних застосунках складається із трьох кроків:

1) розроблення загальної концепції подання;

2) розроблення складових елементів подання;

3) реалізація навігаційної взаємодії елементів подання.

Крок 3.1. Визначення, у якій формі буде відображено сформовані таблиці даних та візуальні структури.

Крок 3.2. Розроблення як подання мультимедійного застосунку на основі програмного забезпечення Microsoft Visual Studio, мови програмування C# за допомогою використання інтерфейсу програмування застосунків Windows Forms.

Крок 3.3. Реалізація навігаційної взаємодії елементів подання.

Навігацію по мультимедійному застосунку з візуалізації інформації мають виконувати, згідно з такою схемою (рис. 3.2).

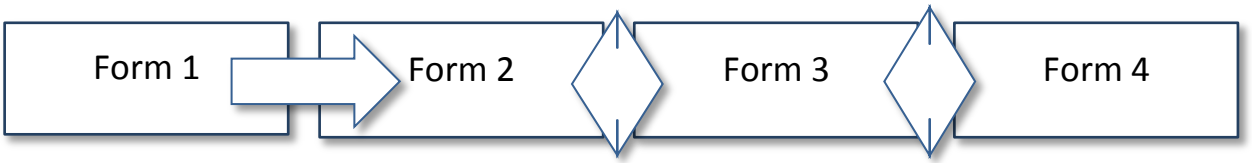


Рис. 3.2. **Навігація по мультимедійному застосунку з візуалізації інформації**

Реалізація навігації елементів візуальної форми відбувається безпосередньо написанням коду для кожної форми та кожного її елементу.

Подання візуальної форми інтерактивно змінює й доповнює візуальні структури, щоб зі статичної презентації дістати візуалізацію шляхом установлення графічних параметрів – дістати подання візуальної структури. Візуальною формою є мультимедійний застосунок, розроблений за допомогою мови програмування C# із використання XML (стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками).

Формування таблиць даних відбувається у два етапи.

Перший етап – формування таблиці даних "терміни – визначення".

Другий етап – формування таблиці даних "термін – фрагмент інформації" на основі використання фрагмента інформації, що візуалізують.

Формування таблиці даних "терміни – фрагменти інформації" полягає у відокремлені фрагмента тексту вихідної інформації, яка містить ключовий термін (рис. 3.3).

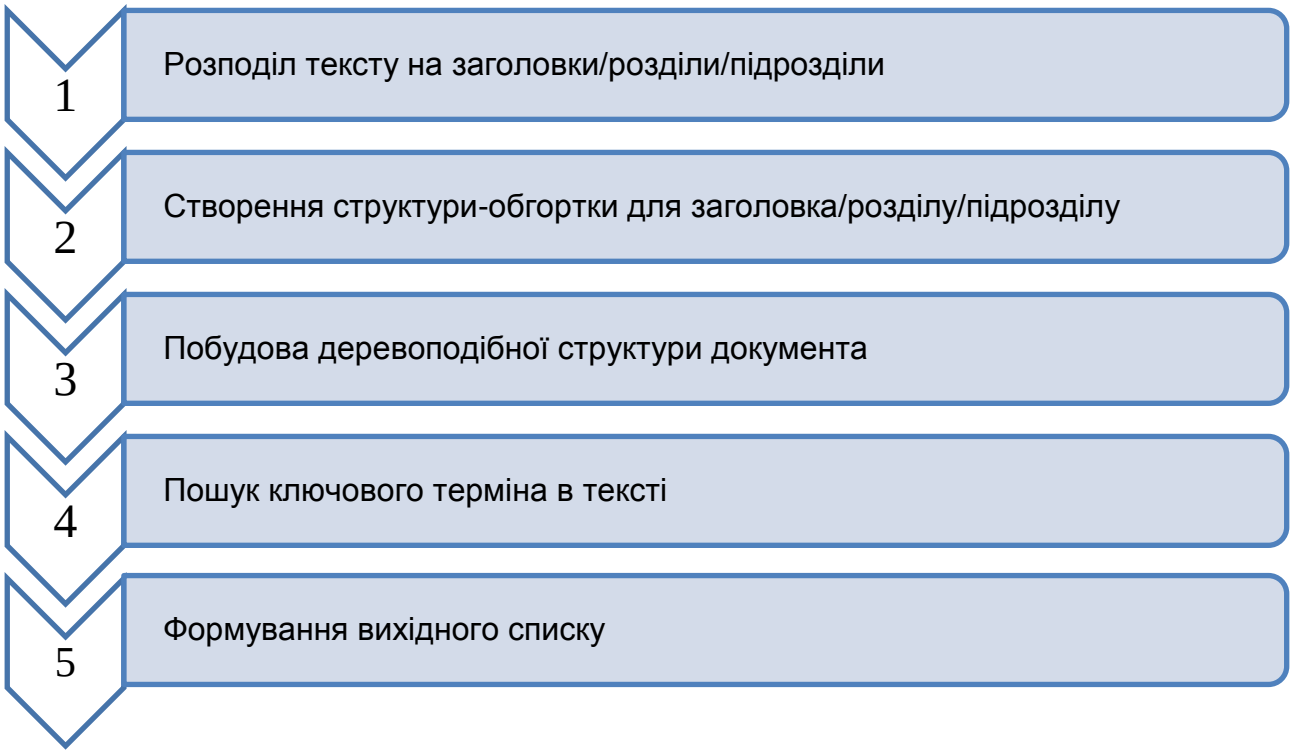


Рис. 3.3. **Алгоритм відокремлення фрагментів тексту інформації, які мають ключові терміни**

Щодо кожного ключового терміна розробляють візуальні структури, які також доповнюють таблиці даних, – це піктограми та зображення.

Для створення візуалізованих табличних даних пропонують використовувати інтерактивну інфографіку.

Задля візуалізації табличних даних у видавничих стандартах пропонують вибрати метод із групи "метафорична візуалізація". Візуальні метафори виконують подвійну функцію: *по-перше*, вони подають інформацію у графічному вигляді, щоб організувати та структурувати її; *по-друге*, вони передають уявлення про подану інформацію через ключові характеристики метафори, яку використовують. Метафори є корисними, оскільки дозволяють пояснити абстрактне поняття в конкретних умовах, що робить його легшим для розуміння. Вони також можуть викликати емоційну реакцію.

Табличні дані може бути візуалізовано за допомогою методу метафоричної візуалізації "параметрична лінійка" (parameter ruler). Цей інтерактивний метод засновано на метафорі логарифмічної лінійки, яка дозволяє користувачам рухати шкали, щоб проаналізувати різні комбінації значень параметрів. "Параметрична лінійка" належить до інтерактивної інфографіки, бо в користувача є можливість взаємодіяти із системою відображення інформації та спостерігати за її реакцією.

Альтернативними методами для вирішення поставлених завдання є індексація і використання алгоритмів класифікації та кластеризації.

Індексація є необхідною для вилучення деяких атрибутів об'єктів візуалізації, що передають сенс і зміст самих об'єктів. Потрібні різні алгоритми індексації, залежно від природи даних: наприклад, алгоритми опрацювання природної мови (словники ключових слів, ключових фраз, групи імен, частини мови) для колекцій текстів; алгоритми опрацювання зображень (сегментація за кольором, яскравістю, структурою) для колекцій зображень; алгоритми опрацювання аудіо (за звуком і висотою тону) для колекцій аудіофайлів; алгоритми опрацювання відео (сегментація по сценах) для відеоколекцій.

Алгоритми класифікації розподіляють об'єкти візуалізації за визначеними категоріями (групами), використовуючи алгоритми машинного навчання, наприклад, алгоритм Баєса, алгоритм k-найближчого сусіда, нейромережеві алгоритми та ін.

Алгоритми кластеризації динамічно розподіляють об'єкти візуалізації на групи шляхом обчислення деякої міри подібності між ними, наприклад, алгоритм k-середнього, самоорганізовані карти Когонена,

ієрархічні алгоритми та ін. У результаті отримують матриці, що описують об'єкти та взаємозв'язки між ними та їхніми групами.

Проте методи індексації та використання алгоритмів класифікації та кластеризації не дають можливостей полегшити сприйняття великої кількості текстової інформації й забезпечити її подання в системах мультимедіа.

3.3. Аналіз особливостей програмної реалізації технології візуалізації інформації для використання в мультимедійних застосунках

Розгляньмо використання методу метафоричної візуалізації "параметрична лінійка" на прикладі візуалізації даних для шрифтового оформлення видань для дітей.

До складу параметрів увійдуть: кегль шрифту, збільшення інтерліньяжу, мінімальна довжина рядка, максимальна довжина рядка, група шрифту, місткість шрифту на кириличній графічній основі, місткість шрифту на латинській графічній основі, накреслення. Горизонтальні шкали з можливими значеннями параметрів, відсортованими від найменшого до найбільшого, мають рухатися відносно одна одної. Інтерактивність будуть реалізовувати завдяки тому, що користувач буде вводити значення кегля, яке його цікавить, і побачить співвідношення значень інших параметрів (табл. 3.2, рис. 3.4).

Таблиця 3.2

Шрифтове оформлення видань для дітей

Кегль шрифту, не менше ніж, пункти	Збільшення інтерльяжу, не менше ніж, пункти	Довжина рядка				Група шрифту	Місткість шрифту, не більше ніж, зн./кв.	Накреслення
		мінімальна		максимальна				
		квадрати	мм	квадрати	мм			
1	2	3	4	5	6	7	8	9
20 і більше	2	6 1/2	117	9 1/2	171	Рубані (гротескові) або із засічками	5,0 (5,5)*	Нормальне або широке світле пряме
16 – 18	4	6 1/2	117	9 1/4	167	Рубані (гротескові) або із засічками	6,0 (6,6)*	Нормальне або широке світле пряме

1	2	3	4	5	6	7	8	9
14	4	6	108	8 1/2	153	Рубані (гротескові)	6,7 (7,4)*	Нормальне або надшироке світле пряме
12**	2	5	90	8 1/2	153	Рубані (гротескові)	7,7	Нормальне, широке або надшироке світле пряме

* Для шрифту на латинській графічній основі.

** Для додаткового тексту, обсяг якого не більше ніж 200 знаків на сторінці.

Рис. 3.4. **Таблиця і результат її візуалізації**

Візуалізовані таким способом таблиці допоможуть спростити процес охоплення великого обсягу інформації, а також скоротити витрати на пошук необхідних даних та закріпити їх на зоровому рівні.

Сформована таблиця даних "термін – визначення – фрагмент стандарту – візуальні структури" має таких набір даних: ключовий термін,

визначення, фрагменти стандарту та візуальні структури. Згруповані дані таблиці, а саме ключові терміни, їхнє визначення та візуальні структури буде подано у вигляді своєрідної бази даних, а саме у вигляді xml-файлу (keywords.xml) (рис. 3.5).

```
<?xml version="1.0" encoding="utf-8" ?>
<keywords>
        <chapter name="...">
            <keyword name="..." image="..." icon="...">...</keyword>
        </chapter>
</keywords>
```

Рис. 3.5. **Структура xml-файлу, який містить елементи таблиці даних "термін – визначення – візуальні структури"**

Структура xml-файлу складається з тегів. У принципі, xml є базою даних із деревоподібною структурою (рис. 3.6).

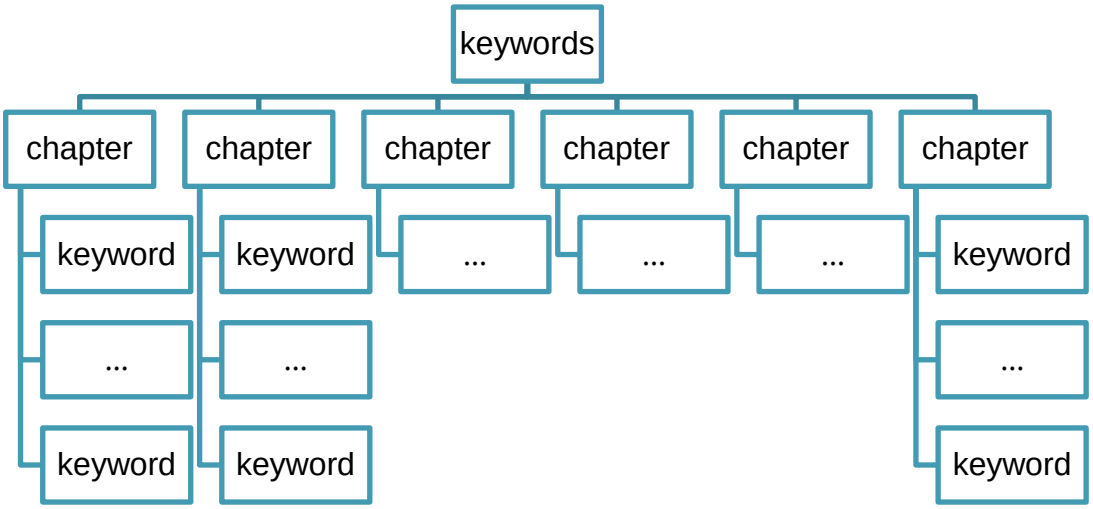


Рис. 3.6. **Деревоподібна структура xml-файлу, який містить елементи таблиці даних "термін – визначення – фрагмент інформації – візуальні структури"**

Кореневий тег <keywords>…</keywords>, у ньому прописують увесь перелік згрупованих ключових термінів. Тег <chapter>… </chapter> становить кожну групу ключових термінів (**загальні поняття, оформлення та виконання сторінки видання, оформлення та виконання блока видання, оформлення та виконання покриття видання, цільове оформлення видання та поліграфічне виконання видання**). Цей тег

має атрибут name="…", у якому у лапках треба оголосити назву самої групи ключових термінів. Тег <keyword>…</keyword> відповідає за збереження елементів із таблиці даних "термін – визначення – візуальні структури", а саме ключовий термін, визначення, візуальну структуру першого та другого типу. Цей тег має три атрибути name="…", image="…", icon="…". Атрибут name="…" відповідає за збереження ключового терміна, назву якого потрібно розмістити в лапках після оголошення атрибуту. Атрибут image="…" відповідає за збереження візуальної структури першого типу, у лапках цього атрибута необхідно прописати шлях збереження відповідного зображення, так само атрибут icon="…" відповідає за збереження візуальної структури другого типу, у лапках цього атрибута необхідно прописати шлях збереження відповідної піктограми. Визначення терміна розміщують між тегами <keyword>…</keyword>.

Сформовану структуру xml-файлу наповнюють елементами з таблиці даних "термін – визначення – візуальні структури" (keywords.xml) (рис. 3.7).

```
<keywords>
      <chapter name="загальні поняття")
      ‹keyword name="Блок видання" image="images\zagall. jpg" icon="images\z2.png"›
       комплект скріплених у корінці друкованих аркушів чи зошитів, що містить усі сторінки майбутнього видання.
       </keyword>
       ‹keyword name="Видання" image="images \zagal.jpg" icon="images\z.png"›
       друкований виріб,  призначений для розповсюдження вміщеної в ньому інформації, пройшов редакційно-видавниче оброблення, поліграфічно самостійно оформлений та має вихідні відомості.
       </keyword>
        ‹keyword name-"Обкладинка" image-"images\zagal2. jpg" icon="images\z3.png">
        зовнішнє покриття видання, що з'єднують із блоком без форзаців (ДСТУ 2068).
       </keyword>
       ‹keyword name="Палітурка" image="images \zagal3.jpg" icon="images\z4.png">
       зовнішнє покриття виробу, що з'єднують з блоком за допомогою двох форзаців і корінцевого матеріалу чи без нього (ДСТУ 2068).
       </keyword>
    </chapter>
</keywords>
```

Рис. 3.7. **Xml-файл, заповнений елементами таблиці даних "термін – визначення – візуальні структури"**

Заповнений елементами таблиці даних xml-файл відповідає за збереження даних, які будуть використано в подальшому розробленні мультимедійного видання.

Структура застосунку "Візуалізація видавничих стандартів" має чотири складові форми, кожна з яких має відповідні елементи (які виконують відповідні навігаційні й інтерактивні завдання) та виконує функціональні завдання. Загальну концепцію роботи мультимедійного застосунку показано на рис. 3.8.

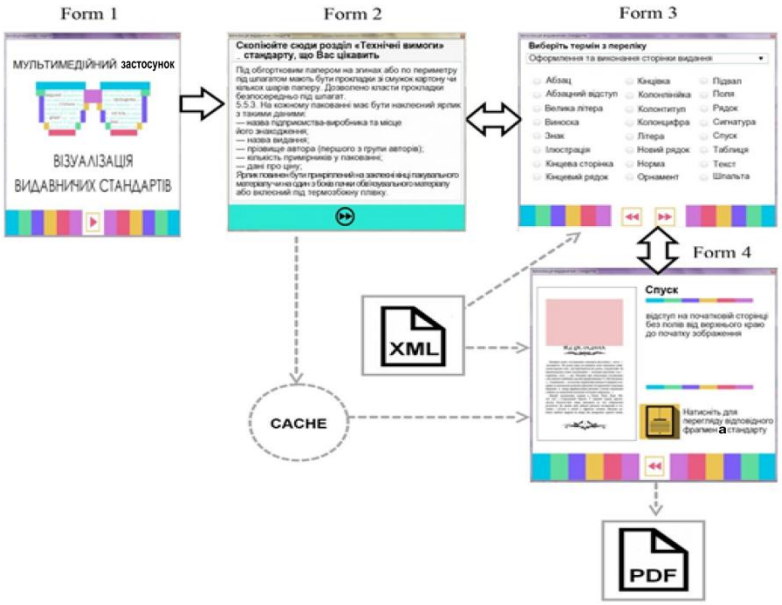


Рис. 3.8. **Схема роботи мультимедійного застосунку "Візуалізація видавничих стандартів"**

Програмна реалізація мультимедійного застосунку починається зі створення нового проєкту в середовищі Microsoft Visual Studio, а саме проєкту WindowsFormsApplication, який дозволить почати створення вікон (form) майбутнього мультимедійного застосунку. На початку створення форм визначмо розміри майбутнього видання та можливість змінювати розміри форми. Для того щоб установити розміри майбутніх форм застосунку потрібно у властивостях форм у розділі Size вказати розміри в пікселях: 900; 754. Майбутній мультимедійний застосунок має зафіксовані розміри форм, тому у властивостях форм у розділі FormBorderStyle вказати значення FixedToolWindow. А також необхідно надати назву майбутньому виданню та закріпити іконку, у розділі Text зазначаймо "Візуалізація інформації", а в розділі Icon визначаймо іконку

для майбутнього мультимедійного застосунку. Для кожної форми розроблено фонове зображення, яке підвантажено за допомогою розділу у властивостях форм BackgroundImage.

Form 1 застосунку має лише один елемент – це навігаційний елемент *кнопка (button1)*, яка відповідає за перехід до наступної форми (рис. 3.9).

Цей код міститься у класі Form1.cs.

```
private void button1_Click(object sender, EventArgs e)
    {
        //Після натиснення кнопки перейти на форму 2
        var form2 = new Form2();
        form2.Show();
        this.Hide ();
    }
```

Рис. 3.9. **Програмний код кнопки, яка здійснює перехід на Form 2**

Form 2 має три елементи: перший *label1* – не має ніяких навігаційних та інтерактивних функцій, це просто напис, який інформує користувача про те, що треба зробити в цій формі, другий елемент – *textBox1*, поле, у яке необхідно скопіювати фрагмент тексту стандарту (розділ "Технічні вимоги") та третій елемент *button1*, що відповідає за збереження тексту, який було введено в textBox1 за допомогою активації відповідного класу, а також за перехід на наступну форму (рис. 3.10).

Цей код міститься у класі Form2.cs.

```
private void button1_Click(object sender, EventArgs e)
        {
            //Запам'ятовує введений текст
            Program.State.Text = textBoxl.Text;

            // Після натиснення кнопки перейти на форму 3
            var form3 = new Form3();
            form3.Show();
            this.Hide();
        }
```

Рис. 3.10. **Програмний код кнопки,**
яка здійснює операцію збереження тексту та переходу на Form 3

Код, який оголошує клас State, який відповідає за збереження введеного тексту міститься у класі Program.cs (рис. 3.11).

```
//Глобальний об'єкт для зберігання інформацію на всіх формах
public static State State  { get; private set; }
```

Рис. 3.11. **Оголошення класу State**

Опис самого класу State міститься у State.cs (рис. 3.12).

```
using System.Collections .Generic;
using System.Linq;
using System.Xml.Ling;

namespace PublicationStandartsVisualization
{
        /// <summary>
       /// Клас, котрий зберігає стан застосунку в час виконання
      /// Також має базу даних ключових термінів
     /// </summary>
     public class State
     {
          public string Text { get; set; }
          public Chapter SelectedChapter { get; set; }
          public Keyword SelectedKeyword { get; set; }
          public IEnumerable‹Chapter> Chapters { get; set; }
          public static State Load()
          {
             //Створення порожнього об'єкта стану
             var result = new State();
             try
             {
               //Відкрити файл із ключовими термінами
               var keywordsDocument = XDocument. Load ("keywords.xml");
               //Опрацювання груп ключових термінів для випадного списку
               result.Chapters = keywordsDocument
               .Descendants("chapter")
               .Select(c => new Chapter
                       {
```

Рис. 3.12. **Програмний код класу State**

```
                    Name = c.Attribute( "name"). Value,
                    Keywords = c.Elements("keyword")
                    .Select(ParseKeyword).ToArray ()
                    }) .ToArray();
        }
            catch
            {
             }
              return result;
        }
        //Опрацювання кожного ключового терміну з конфігураційного файлу
        private static Keyword ParseKeyword(XElement el)
        {
            var imageAttr = el.Attribute ("'image");
            var iconAttr = el.Attribute("icon");
            return new Keyword
                {
                    Name = el.Attribute("name"). Value,
                    Image = imageAttr != null ? imageAttr.Value : null,
                    Icon = iconAttr != null ? iconAttr.Value : null,
                    Description = el. Value
                 };
            }
        }
    }
```

Закінчення рис. 3.12

Form 3 має п'ять елементів: перший *label1* – не має ніяких навігаційних та інтерактивних функцій, це просто напис, який інформує користувача про те, що треба зробити в цій формі; другий елемент *comboBox1* – це випадний перелік назв груп термінів; третій елемент *flowLayoutPanel1* – відображає перелік ключових термінів за допомогою елементів RadioButton; четвертий елемент *button1* – перехід на Form 2; п'ятий елемент *button2* – перехід на Form 3.

Другий і третій елемент пов'язано один з одним, тому що під час вибору одного з варіантів назв груп, поданих у comboBox1, будуть змінювати набір ключових термінів, які містяться у складі елементу flowLayoutPanel1 (рис. 3.13).

Цей код міститься у класі Form3.cs.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
      // Перевірка чи можна опрацьовувати події від випадного списку
      if (lisInitialized) return;

      //Очищення панелі з RadioButton
      flowLayoutPanel1.Controls.Clear();

      // Запам'ятовує вибраний пункт зі списку
      var chapter = comboBox1.SelectedItem as Chapter;
      if (chapter == null) return;
      Program.State.SelectedChapter = chapter;
     // Заповнення панелі з RadioButton зі списку ключових термінів, пов'яза-
ного з пунктом із випадного списку
     bool hasSelection = false;
     foreach (var k in chapter.Keywords)
     {
       // Створення RadioButton із назвою ключового терміна
       var rb = new RadioButton {Text = k.Name, Margin = new Padding(5), Tag = k,
       Width = 300, Height=28);
       // Якщо це останній вибраний ключовий термін (користувач повернувся
на форму з форми 4), позначити його вибраним
       if (Program.State.SelectedKeyword == k)
       {
          hasSelection = true;
          rb.Checked = true;
       }
         flowLayoutPanel1.Controls.Add(rb);
       }
      // Якщо не вибрано жодного ключового терміна, вибрати перший у списку
      if (!hasSelection)
     (flowLayoutPanel1.Controls[0] as RadioButton).Checked = true;
      }
}
```

Рис. 3.13. **Програмний код елементів comboBox1 (flowLayoutPanel1)**

Кнопки button1 та button2 здійснюють навігацію – перехід на Form 2 та Form 4 (код також міститься у файлі Form3.cs) (рис. 3.14).

```
private void button1 _click(object sender, EventArgs e)
   {
       // Після натиснення кнопки перейти на форму 2
       var form2 = new Form2();
       form2.Show();
       this.Hide;
    }
private void button2_Click(object sender, EventArgs e)
   {
          // Пошук вибраного RadioButton і запам'ятовування пов'язаного ключового терміна
              foreach (var rb in
flowLayoutPanel1.Controls.OfType<RadioButton>())
           if (rb.Checked)
           {
           Program.State.SelectedKeyword = rb.Tag as Keyword;
           }
          // Після натиснення кнопки перейти на форму 4
          var form4 = new Form4();
          form4.Show();
          this.Hide;
```

Рис. 3.14. **Програмний код для *button1 ma button2***

Form 4 має шість елементів: *label 1* – відображення ключового терміна, *piturebox* – відображення першого типу візуальних структур; *richtextbox* – відображення визначення ключового терміну; *button1* – перехід на Form 3; *button 2* – відкриття pdf-файлу з візуальної структурою другого типу; *label 2* – напис, який відповідає завданню Button 2 (рис. 3.15).

Цей код міститься у класі Form 4.cs.

```
public Form4()
{
    InitializeComponent ();
```

Рис. 3.15. **Програмний код для Form 4**

```
            //Ключовий термін, вибраний на Form3
             var keyword = Program.State.SelectedKeyword;
             if (keyword != null)
             {
               //Відображення зображення, пов'язаного із ключовим терміном
               //Якщо зображення немає, показати зображення за замовчуванням
                 if (string.IsNul10rEmpty (keyword. Image))
                 {
                  pictureBox1.Image = Properties.Resources.no_image;
                 }
                 else
                 {
                     pictureBox1. Image = new Bitmap(keyword. Image);
               }
                //Відобразити піктограму, пов'язану із ключовим терміном на кнопці
виведення PDF-файлу
             //Якщо піктограми немає, показати піктограму за замовчуванням
             if (string.IsNul10rEmpty (keyword.Icon))
             {
                   button2.Image = Properties.Resources.no_icon;
              }
              else
              {
                    button2.Image = new Bitmap(keyword.Icon);
               }
               //Виведення назви ключового терміна
               label1.Text = keyword. Name;

               //Виведення опису ключевого терміна
               richTextBox1.Clear();
               richTextBox1.SelectAl1();
               var i - richTextBox1.SelectionLength;
               richTextBox1.AppendText((keyword.Description ?? "").Trim(new[] {
' ',  '\r ',  '\n' , '\t'  } ));
             }
         }
          private void button2_Click(object sender, EventArgs e)
         {
```

Продовження рис. 3.15

```
            //Пошук у тексті, уведеному до Form 2, ключового терміна, вибраного
на Form 3
                        var chapters - TextSearch.FindStrings (Program.State. Text,
Program.State.SelectedKeyword.Name).ToArray();
            //Якщо нічого не знайдено, показати повідомлення
            if (chapters.Length == 0)
            {
              MessageBox.Show("Відповідні фрагменти СОУ не знайдено!");return;
            }
            //Створення тимчасового файлу для запису PDF-документа
            var file = Path. GetTempFileName ();
            file = Path.ChangeExtension(file,"pdf");

            // Генерація PDF-документ у тимчасовому файлі
            PDFHelper.CreatePdf(file, chapters);
            //Дочекатися завершення операції
            System.Threading.Thread.Sleep (1800);
            //Відкрити PDF-документ
            Process.Start (file);
        }
        private void button1_Click(object sender, EventArgs e)
        {
                // Після натиснення  кнопки перейти на форму 3
                var form3 - new Form3();
                 form3.Show();
                 this.Hide();
         }
```

Закінчення рис. 3.15

У процесі написання програмного коду для Form 4 було додано два допоміжні класи – TextSearch.cs, який виконує пошук фрагментів тексту, який було введено до Form 2, у якому міститься ключовий термін, який вибирають на Form 3, та PDFHelper.cs, який генерує PDF-документ.

Структура застосунку "СОУ 22.2-02477019-11:2014 Поліграфія. Видання для дітей. Загальні технічні вимоги" має вісім складових форм, кожна з яких має відповідні елементи (які виконують відповідні навігаційні й інтерактивні завдання) та виконує функціональні завдання. Загальну концепцію роботи мультимедійного застосунку показано на рис. 3.16.

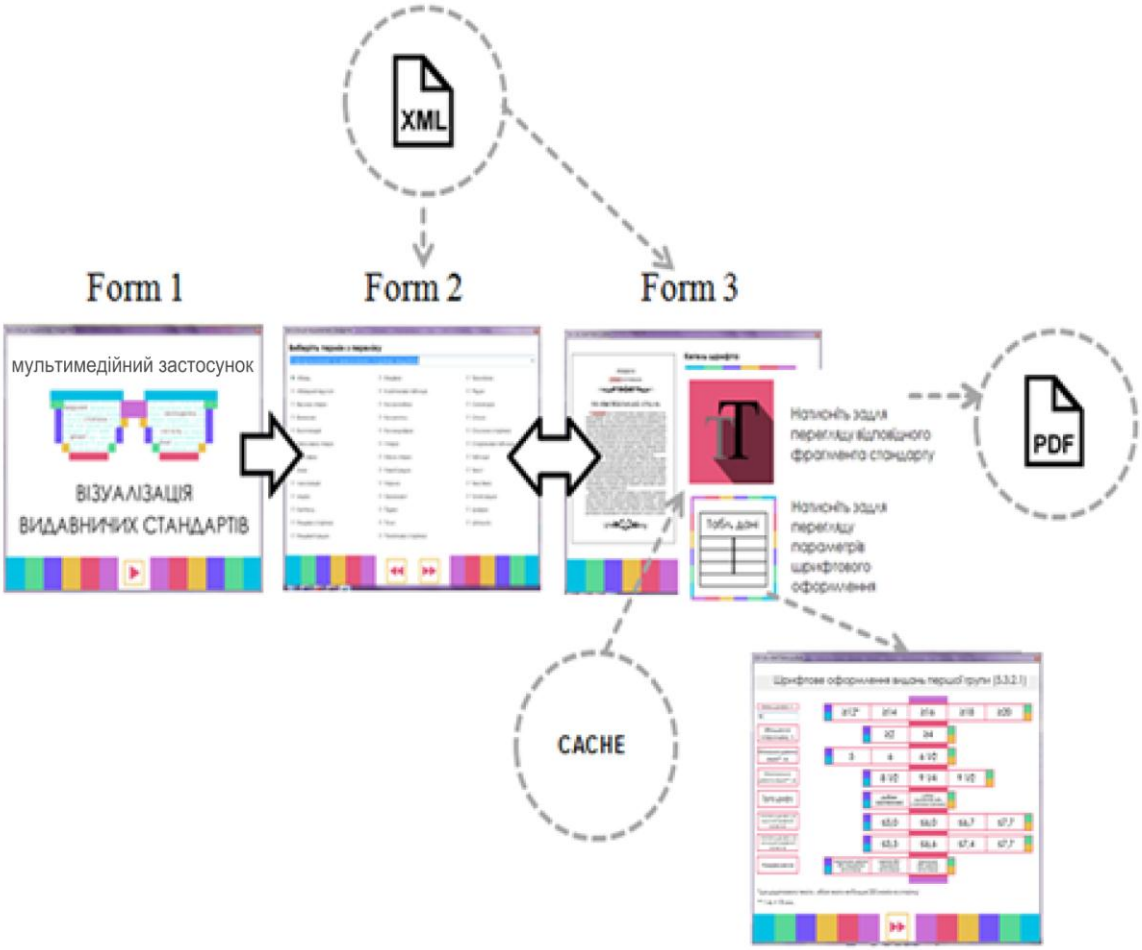


Рис. 3.16. **Схема роботи мультимедійного застосунку "Візуалізація видавничих стандартів"**

⚠ Загалом функціонал залишився з попереднього застосунку, але новий доповнився кнопкою, яка відкриває вікно з візуалізованими таблицями даних. Ця кнопка з'являється в той момент, коли користувач вибрав хоча б один ключовий термін, який міститься у складі табличних даних. Щодо функціональних можливостей було додано елемент textBox, який дозволяє користувачу вводити значення параметра кегль шрифту й, залежно від того, яке значення було введено, змінюється співвідношення інших параметрів щодо введеного значення. Елемент також реалізує інтерактивність мультимедійного застосунку, залежно від уведеного параметра кегля шрифту, змінюється вид візуалізованих таблиць даних.

Код елементу textbox показано на рис. 3.17.

```
private void textBox1_TextChanged(object sender, EventArgs e)
        { try
            {
                if (double.Parse (textBox1.Text) < 14 &&
 double.Parse(textBoxl.Text) ›= 12)
            {
                pictureBox1.Image =
PublicationStandartsVisualization.Properties.Resources._3_1;
             }
             if (double.Parse (textBox1.Text) < 16 &&
double.Parse(textBoxl.Text) >= 14)
             {
                    Clear();
                    pictureBox1.Image =
PublicationStandartsVisualization.Properties.Resources._3_2;
              }
              If (double.Parse(textBox1.Text) < 18 &&
double.Parse(textBoxl.Text) >= 16)
              {
              Clear();
              pictureBoxl.Image =
PublicationStandartsVisualization.Properties.Resources._3_3;
              }
             If (double.Parse(textBox1.Text) < 20 &&
double.Parse(textBoxl.Text) >= 18)
             {
              Clear();
              pictureBoxl.Image =
PublicationStandartsVisualization.Properties.Resources._4_2;
             }
             else
              {
                    if (double.Parse(textBoxl.Text) ›- 20)
                     {
                         Clear();
                        pictureBox1.Image
PublicationStandartsVisualization.Properties.Resources._4_3
```

Рис. 3.17. **Програмний код для елементу *textbox***

Результатом здійсненої програмної реалізації є готові мультимедійні застосунки, які, згідно з методикою візуалізації стандартів щодо

додрукарської підготовки видань, є кінцевим продуктом третього етапу подання візуальної форми.

Ще одним прикладом реалізації методу метафоричної візуалізації "параметрична лінійка" є мультимедійний дидактичний комплекс із навчальної дисципліни "Технологічні процеси видавничо-поліграфічної справи" (ТП ВПС), розроблений авторами в межах системи e-learning (рис. 3.18).

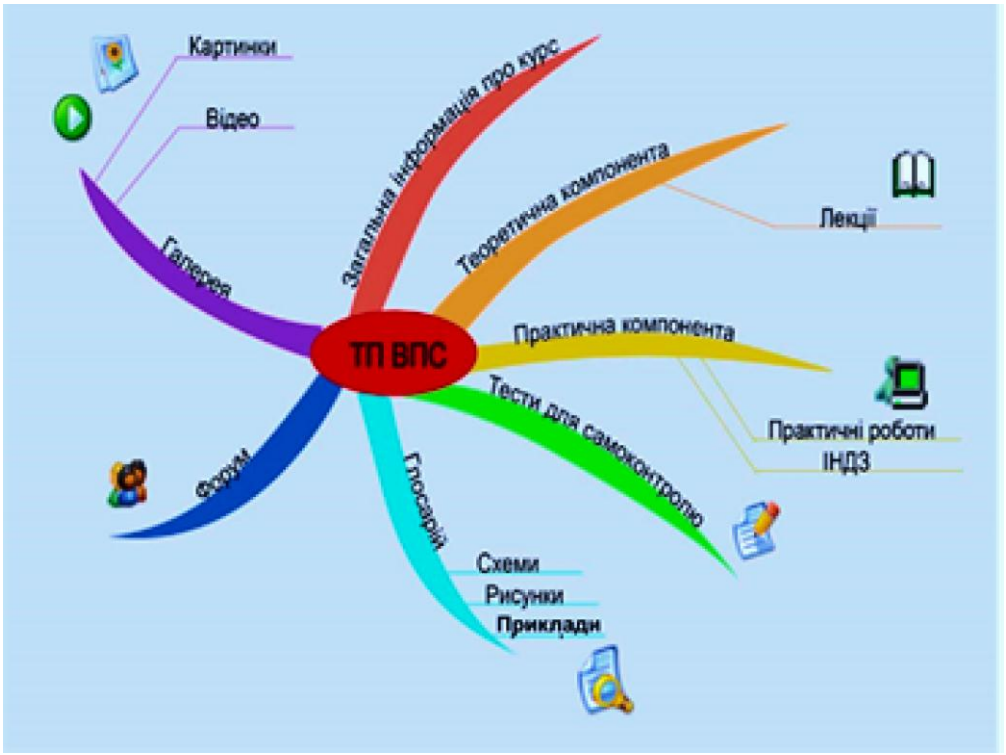


Рис. 3.18. **Загальна структура мультимедійного дидактичного комплексу з навчальної дисципліни "Технологічні процеси видавничо-поліграфічної справи" (ТП ВПС)**

Для створеного мультимедійного дидактичного комплексу було використано такі типи об'єктів інтерактивної інфографіки:

інфографічні розповіді для побудови цікавих і наочних практичних занять;

інтерактивні тестові системи у вигляді інфографічних таблиць;

інтерактивні картосхеми для організації й адміністрування навчального процесу, складання розкладу, обліку студентів і їхньої успішності;

інфографічні таблиці, які дають можливість студентам виконувати спільні завдання чи радитися під час їхнього виконання.

 Можливими обмеженнями використання цієї технології візуалізації інформації для використання в мультимедійних застосунках є індивідуальні особливості розумової діяльності користувача, сформовані під впливом особистісних і ситуаційних чинників. Зокрема, розглянуту технологію візуалізації може бути ефективно використано користувачами з наочно-дійовим і наочно-образним типами мислення. Але об'єктивні труднощі з використанням технології візуалізації можуть виникнути в користувачів зі словесно-логічним типом мислення.

Практична складова до підрозділу 3
Проєктування інтерфейсу користувача

Мета – вивчення основних рекомендацій щодо проєктування шару подання, освоєння процесу проєктування інтерфейсу користувача (UI) у типовій архітектурі багатошарового застосування мобільних **застосунків** та набуття навичок у розробленні компонентів шару подання багатошарової архітектури.

Теоретичні засади

Рекомендовано на перших етапах процесу розроблення додавати питання, пов'язані з дослідженням зручності використання, опитування та інтерв'ю, щоб зрозуміти, що користувачі хочуть та очікують від мобільних застосунків, і на підставі цих відомостей проєктувати ефективний UI [4].

Модель шару подання (завд. 1)

Шар подання мобільних застосунків відповідає за реалізацію та відображення інтерфейсу користувача, а також управління взаємодією з користувачем. Цей шар, крім компонентів, що організовують взаємодію з користувачем, містить елементи управління для введення даних та їхнього відображення.

На рис. 3.19 показано місце шару подання в загальній архітектурі мобільних застосунків.

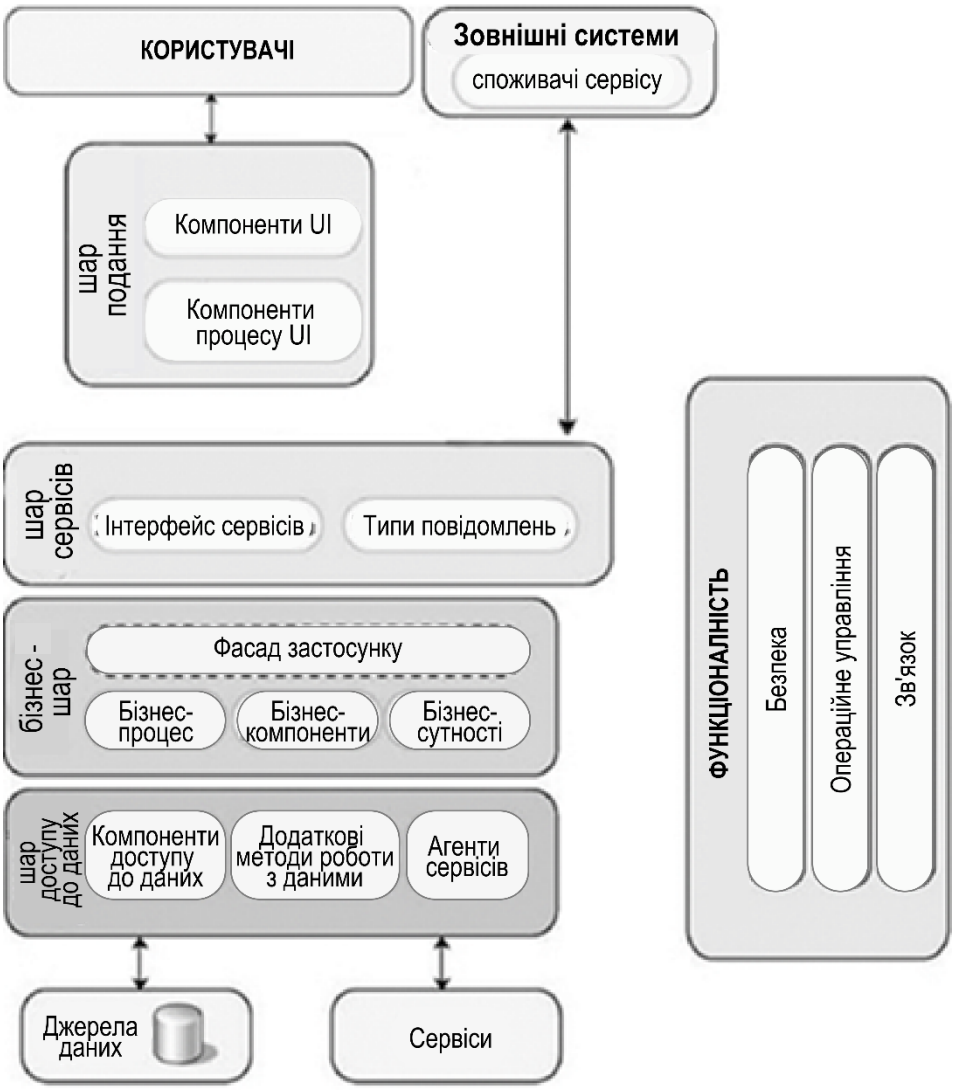


Рис. 3.19. **Шар подання в типовому застосунку**

Шар подання мобільних застосунків зазвичай містить такі компоненти [2]:

• *Компоненти інтерфейсу користувача.* Це візуальні елементи програми, які використовують для відображення даних користувачеві та приймання введення користувача.

• *Компоненти логіки подання.* Логіку подання реалізують кодом програми, що визначає поведінку та структуру програми та залежить від конкретної реалізації інтерфейсу користувача.

Під час реалізації шаблона Separated Presentation у мобільних застосунках можуть використовувати такі компоненти логіки подання: презентатор (Presenter), модель презентації (Presentation Model) та модель подання (View Model).

Шар подання мобільних застосунків також може містити компоненти моделі шару подання (Presentation Layer Model), які інкапсулюють дані бізнес-шару, або компоненти сутності подання (Presentation Entity), які інкапсулюють бізнес-логіку та дані у формі, зручній для використання шаром подання.

Для того щоб створювана реалізація UI відповідала вимогам до програмного забезпечення мобільних застосунків, рекомендовано дотримуватися кращих практик і керуватися такими принципами [2; 3]:

• Вибирайте тип програми, що відповідає цілям та вирішуваним завданням. Від вибраного типу програмного забезпечення мобільних застосунків будуть істотно залежати доступні варіанти реалізації шару подання. Визначтеся, чи будете реалізовувати насичений клієнт, вебклієнт або насичений інтернет-застосунок (rich Internet application, RIA). Рішення мають ухвалювати на підставі вимог, що становлять до застосунку, та обмежень, що накладають організація чи середовище.

• Вибирайте відповідну технологію UI. Різні типи застосунків забезпечують різні набори технологій розроблення шару подання. Кожен тип технології має індивідуальні переваги, які визначають можливість створення відповідного шару подання.

• Використовуйте відповідні шаблони. Шаблони шару подання пропонують перевірені розв'язання звичайних проблем, що виникають під час проєктування шару подання. Пам'ятайте, що не всі шаблони є застосовними однаково до всіх архетипів, але загальний шаблон Separated Presentation, у якому аспекти, що стосуються подання, відокремлено від базової логіки програми, підходить для всіх типів програм. Спеціальні шаблони, як-от MVC, MVP та Supervising Presenter, зазвичай використовують у шарі подання насичених клієнтських програм та RIA. Різновиди шаблонів MVC та MVP можна застосовувати у вебзастосунках.

• Розділяйте функціональні сфери. Використовуйте спеціальні компоненти UI для формування візуального подання, відображення та взаємодії з користувачем. У складних випадках, або якщо хочете забезпечити можливість модульного тестування, зверніть увагу на спеціальні компоненти логіки подання для управління опрацювання взаємодії з користувачем.

• Ураховуйте рекомендації щодо проєктування інтерфейсу користувача, включно з такими аспектами, як зручність і простота доступу, локалізація та зручність використання. Для вибраних типів клієнта та технологій ознайомтеся зі встановленими рекомендаціями щодо інтерактивності UI, зручності використання, сумісності із системою, відповідності вимогам, а також із шаблонами проєктування UI та застосуйте ті з них, які відповідають дизайну та вимогам створюваної програми.

• Дотримуйтеся принципів орієнтованого користувача проєктування. Перш ніж розпочинати проєктування шару подання, зрозумійте свого споживача. Використовуйте опитування, дослідження зручності використання та інтерв'ю для вибору варіанта інтерфейсу користувача, що найбільш відповідає вимогам замовника.

• Опрацювання тривалих запитів мають реалізовувати з урахуванням часу відгуку користувача, а також зручності обслуговування та тестування коду. Під час проєктування опрацювання запитів керуйтеся такими рекомендаціями:

використовуйте асинхронні операції або фонові потоки, щоб уникнути блокування UI під час тривалих дій у програмах. Застосовуйте AJAX для асинхронних запитів в ASP.NET. Забезпечуйте користувачеві зворотний зв'язок про хід виконання процесу тривалих операцій. Надайте користувачеві можливість скасувати тривалу операцію;

уникайте змішування логіки опрацювання UI та формування візуального подання;

під час виконання ресурсомістких викликів до віддалених ресурсів або шарів, як-от виклик вебсервісів або запит до бази даних, здійснити аналіз, можливо, буде доцільним.

Виконуйте ці запити до сервісу з детальним інтерфейсом (багато дрібних запитів) або з узагальненим інтерфейсом (один великий запит). Якщо користувач запитує великі обсяги даних для виконання операції, спочатку вилучайте лише те, що необхідно для відображення та початку роботи, а потім поступово довантажуйте дані у фоновому потоці або в міру необхідності (наприклад, розподіл даних на сторінки та віртуалізація UI).

• Забезпечте зрозумілі користувачеві повідомлення про помилки для сповіщення про помилки програми, але переконайтеся, що не вмикаєте конфіденційні дані до сторінок помилок, у повідомлення про помилки,

файли журналів. Намагайтеся привести програму до погодженого стану або, якщо це неможливо, забезпечте завершення виконання.

• Розробляйте стратегію навігації так, щоб користувачі могли зручно переміщатися екранами або сторінками програми та можна було відокремити функціональність навігації від формування та опрацювання UI.

• Забезпечте одноманітне подання навігаційних посилань та елементів управління в усьому застосунку, щоб не заплутувати користувача і приховати складність програми. Під час проєктування стратегії навігації керуйтеся такими рекомендаціями:

проєктуйте панелі інструментів та меню, щоб користувачі могли швидко знаходити функції, що надають UI;

забезпечте передбачуваність реалізації навігації між формами за допомогою майстрів і визначтеся з тим, як будете зберігати стан навігації між сеансами в разі потреби;

уникайте дублювання логіки для обробників подій навігації та по можливості не вказуйте в коді шляху переходів, для опрацювання звичайних дій із багатьох джерел використовуйте шаблон Command (Команда).

Застосування компонентів на шарі подання (завд. 2)

Загальні рекомендації щодо проєктування компонентів мобільних застосунків містять відомі принципи проєктування SOLID, рекомендують проєктувати дуже зв'язані компоненти, компонент не має залежати від внутрішніх деталей інших компонентів та від того, як компоненти будуть взаємодіяти один з одним.

Для шару подання мобільних застосунків реалізують два типи компонентів: безпосередньо інтерфейсу користувача та логіки подання.

Розгляньмо їх більш докладно:

• *Компоненти інтерфейсу користувача*.

У цьому разі конкретну реалізацію інтерфейсу користувача програми інкапсульовано в компоненти інтерфейсу користувача мобільних застосунків. Це візуальні елементи програми, які використовують для відображення даних користувачеві та приймання введення користувача. Під час реалізації шаблона Separated Presentation ці компоненти називають поданнями (Views), і їхня роль полягає в наданні користувачеві інтерфейсу.

Інтерфейс забезпечує найбільш відповідне подання даних і логіки програми, а також в інтерпретації введення користувача й передавання його в компоненти *логіки подання*, які визначають вплив уведення на дані та стан програми. У деяких випадках у компонентах інтерфейсу користувача може міститися спеціальна логіка реалізації інтерфейсу користувача, однак, зазвичай, вони містять мінімальний обсяг логіки програми, оскільки це може негативно позначитися на зручності обслуговування та можливості повторного використання, а також ускладнити модульне тестування.

Стратегії опрацювання помилок та валідації (завд. 3)

Компоненти UI є зовнішньою межею програми мобільних застосунків і тому мають реалізовувати відповідну стратегію опрацювання помилок для забезпечення стабільності програми та позитивного враження під час взаємодії з користувачем.

Під час проєктування стратегії опрацювання помилок розгляньте такі варіанти [2]:

• Стратегія централізованої опрацювання винятків. Опрацювання винятків та помилок належить до наскрізної функціональності. Її мають реалізовувати в окремих компонентах, які забезпечують централізацію цієї функціональності та роблять її доступною у всіх шарах програми. Це також спрощує обслуговування та сприяє повторному використанню.

• Протоколування винятків. Дуже важливо протоколювати помилки в межах системи, щоб служба підтримання могла виявляти та діагностувати їх. Це важливо для компонентів подання, але може створювати великі труднощі для коду, який виконують на клієнтському боці. Будьте обережними та ретельно вибирайте методи протоколювання інформації особистого порядку (Personally Identifiable Information, PII) або конфіденційних даних і зверніть особливу увагу на розмір та розміщення журналу.

• Виведення на екран зрозумілих користувачеві повідомлень. У разі використання цієї стратегії під час виникнення помилки на екран виводять зрозуміле користувачеві повідомлення із зазначенням причини помилки й описом, як її можна виправити. Наприклад, помилки перевірки даних мають відображати так, щоб було зрозуміло, які дані помилкові

та чому. У цьому повідомленні також зазначено, як користувач може виправити або ввести дійсні дані.

• Вирішення повторної спроби. У разі використання цієї стратегії на екран виводять зрозуміле користувачеві повідомлення, що пояснює причину помилки та пропонує повторити операцію. Така стратегія є корисною, якщо помилки виникають через настання тимчасових винняткових ситуацій, як-от відсутність ресурсу або закінчення часу очікування мережі.

• Виведення на екран універсальних повідомлень. Якщо у програмі виникає непередбачена помилка, ці помилки необхідно запротоколювати, але для користувача вивести лише універсальне повідомлення. Надайте користувачеві унікальний код помилки, який може бути подано групі технічного підтримання. Ця стратегія є корисною в разі непередбачених винятків. Зазвичай, у разі виникнення непередбаченого вимикання рекомендовано закрити програму, щоб запобігти пошкодженню даних або ризикам безпеки.

Ефективна стратегія валідації введення користувача допоможе фільтрувати небажані або зловмисні дані та буде сприяти підвищенню захищеності програми. Зазвичай, валідацію введення здійснюють шаром подання, тоді як перевірку на відповідність бізнес-правил виконують компонентами бізнес-шару.

Під час проєктування стратегії перевірки, насамперед, необхідно визначити всі дані, що підлягають перевірці. Наприклад, уведення від вебклієнта в поля форми, параметри (наприклад, дані операцій GET і POST та рядки запитів), приховані поля і стан подання (view state) підлягають перевірці. Загалом перевіряти мають всі дані, що надходять із джерел, які не мають довіри.

Для застосунків, що мають компоненти й на боці клієнта, і на боці сервера, як-от RIA або насичені клієнтські застосунки, що викликають сервіси на сервері застосунків, крім усіх перевірок на клієнті, мають виконувати додаткову перевірку на сервері. Але для підвищення зручності використання та з міркувань продуктивності деякі з перевірок можна продублювати на клієнта. Перевірка клієнта дозволяє забезпечувати користувачам швидкий зворотний зв'язок у разі введення ними некоректних даних. Це може зберегти час і смугу пропускання, але не забувайте, що зловмисники можуть обійти будь-яку реалізовану на клієнті перевірку.

Практичні завдання

Завдання 1. Розробіть попередню модель шару подання мобільних **застосунків** з урахуванням побажань користувача, виберіть відповідний вихідним даним тип програми та технології UI.

Завдання 2. Запропонуйте пропозиції щодо застосування компонентів на цьому шарі програми, що розробляють. Перелічіть, які функції ці компоненти мають виконувати, із метою досягнення необхідної функціональності.

Завдання 3. Розробіть пропозиції щодо реалізації за допомогою компонентів стратегій опрацювання винятків та валідації.

Контрольні запитання для самоперевірки

1. Назвіть основні завдання візуалізації інформації в мобільних застосунках.
2. Перелічіть загальні рекомендації щодо проєктування інтерфейсу користувача.
3. Опишіть технологію візуалізації інформації в мобільних застосунках.
4. Укажіть свою думку про застосування принципів SOLID під час проєктування класів, що входять до компоненту інтерфейсу користувача.
5. Опишіть основний зміст методики проєктування компонентів подання.
6. Проаналізуйте специфіку програмної реалізації технології візуалізації інформації в мобільних застосунках.

Рекомендована література: [1; 3; 6; 8; 15; 17].

Розділ 2
Основи програмування мобільних застосунків

4. Основи розроблення програм для операційної системи Android

Мета – ознайомлення студентів з основами розроблення програм для операційної системи Android.

Професійні компетентності: здатність аналізувати структуру та контент проєктів інтерактивних медіа; здатність оцінювати та забезпечувати якість виконуваних робіт.

4.1. Поняття Android SDK. Менеджер із пакетів Android SDK

 **Android SDK** – це набір інструментів, які дозволяють розробляти програми для платформи Android. До його складу входять бібліотеки, що надають програмісту API платформи Android, утиліти для створення та збирання застосунків, управління образами віртуальних пристроїв і виконання інших дій. Android SDK розповсюджують безкоштовно для всіх основних платформ і його може бути завантажено із сайта Google: http: //developer.android.com/sdk/index.html.

Android SDK не містить компілятора мови Java, тому для компіляції застосунків Android необхідно також набір інструментів Java Development Kit (JDK). Останню версію JDK від Oracle може бути завантажено із сайта Oracle за адресою: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Для спрощення розроблення може бути використано одне з інтегрованих середовищ (Intergrated Development Environment, IDE). Нині підтримання розроблення під Android є у всіх основних середовищах розроблення для Java, зокрема IntelliJ IDEA, NetBeans та Eclipse. Слід зазначити, що наявність IDE не є обов'язковою для розроблення, оскільки всі необхідні для збирання та розгортання програми операції може бути виконано засобами командного рядка. Цей посібник не передбачає використання будь-якої конкретної IDE.

 Android SDK підтримує розроблення під всі офіційні версії платформи Android із використанням безлічі бібліотек, включно з багатьма бібліотеками сторонніх розробників. Для управління пакетами, що забезпечують розроблення з використанням цих бібліотек, використовують утиліту Android. Її можна знайти в підкаталозі інструментів усередині каталога SDK.

Утиліта Android підтримує як інтерфейс командного рядка, так і графічний інтерфейс користувача. Для управління бібліотеками доцільно використовувати графічний інтерфейс, доступ до якого можна здобути, запустивши утиліту Android без параметрів. Після старту на екрані з'явиться список пакетів, що містить як установлені в системі пакети, так і пакети, доступні для встановлення. Для встановлення необхідних пакетів необхідно вибрати їх у загальному списку та натиснути кнопку Install packages. Після прийняття ліцензії почнеться завантаження вибраних пакетів з інтернету та їхнє встановлення.

Пакети зі списку згруповано за версіями платформи Android. Кожна версія платформи має подвійну нумерацію: комерційну та внутрішню: наприклад, Android 4.0.3 відповідає API 15. Перший тип нумерації використовують для позначення підтримання можливостей платформи пристроями на ринку, тоді як для процесу розроблення застосунків повсюдно використовують другий тип нумерації.

 У кожній із груп у списку містяться такі елементи: основний набір API для розроблення застосунків під цю платформу (SDK Platform); приклади застосунків (Samples for SDK); документацію на API (Documentation for Android SDK); вихідні тексти бібліотек платформи (Sources для Android SDK); образи віртуальних пристроїв для різних архітектур (ARM EABI v7a System Image, Intel x86 Atom System Image та ін). Крім того, у списку може бути наведено бібліотеки сторонніх розробників, зокрема призначені для певного класу пристроїв (наприклад, Google TV Addon).

Щоб розпочати розроблення, необхідно встановити основний набір бібліотек (SDK Platform), образи віртуальних пристроїв (System Images) для конкретної платформи, а також загальний набір інструментів, не прив'язаний до версії API (Android SDK Tools, Android SDK Platform-tools у групі Tools). Інші пакети є необов'язковими.

4.2. Створення проєкту

Створити шаблон для нового проєкту можна засобами утиліти Android, описаної в попередньому підрозділі.

Для цього використовують інтерфейс командного рядка, наприклад:

```
android create project --target android-15 --name HelloWorld \
    --path HelloWorld --activity MainActivity \
```

 Рекомендовано додавати шляхи до підкаталогів tools і platform-tools каталога, що містить SDK, у змінну оточення PATH. Це дозволить викликати утиліту Android та інші інструменти командного рядка без вказівки повного шляху так, як це зроблено в наведеному прикладі. --package ru.ac.uniyar.helloworld.

Призначення ключів наведеної команди такі:

• --target – ідентифікатор платформи, для якої створюють проєкт (список усіх доступних платформ визначено набором установлених пакетів Android SDK і може бути визначено за допомогою команди android list target);

• --name – назва створюваної програми;

• --path – шлях до каталога створюваного проєкту;

• --activity – назва головної активності проєкту, що створюють (відповідає головному екранові програми; більш докладно активності розглянуто в розділі 2);

• --pakage – назва кореневого пакета створюваної програми (усі класи створюваної програми буде розміщено всередині цього пакета).

У результаті виконання команди з попереднього пункту в поточному каталозі буде створено каталог проєкту з назвою HelloWorld, що містить такі файли та каталоги:

• AndroidManifest.xml – файл маніфесту;

• build.xml – складальний файл ant;

• ant.properties, project.properties, local.properties – файли конфігурації ant;

• src – каталог, що містить вихідні тексти програми;

• res – каталог, який містить файли ресурсів.

 Файл AndroidManifest.xml є головним конфігураційним файлом Android-програми. Він містить визначення всіх компонентів, із яких складається застосунок, описує способи взаємодії між ними, умови складання

проєкту, права доступу до різних ресурсів тощо. Далі на рис. 4.1 показано приклад найпростішого файлу маніфесту й описано його основні компоненти.

```
<?xml version="1.0" encoding="utf 8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="ru.ac.uniyar.helloworld"
android:versionCode="1" android:versionName="1.0">
<uses−sdk android:minSdkVersion="15" />
<uses−feature android:name="android.hardware.location" />
<uses permission
android:name="android.permission.ACCESS COARSE LOCATION"/>
<uses permission android:name="android.permission.INTERNET"/>
<application android:label="@string/app name"
android:icon="@drawable/ic launcher">
<activity android:name="MainActivity" android:label="@string/app name">
<intent  filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent  filter>
</activity>
</application>
</manifes
```

Рис. 4.1. **Приклад найпростішого файлу маніфесту**

Кореневим елементом файлу маніфесту є елемент <manifest> із такими атрибутами: pakage (назва кореневого пакета застосунку), android:versionName (номер версії програми, який може бачити користувач), android:versionCode (внутрішній номер версії програми), що використовують під час оновлення програм, установлених на пристрої).

Елемент <uses-sdk> визначає версії SDK, які може бути використано для складання проєкту. У прикладі раніше за допомогою атрибута android:minSdkVersion указано, що проєкт може бути складено

за допомогою SDK версії API не нижчою за 15, яка відповідає Android 4.0.3. За спроби складання проєкту за допомогою невідповідної версії SDK буде видано повідомлення про помилку.

Елементи <uses-feature> використовують для декларації можливостей, які має підтримувати цільовий пристрій для правильного функціонування програми. Наприклад, значення android.hardware.location

означає, що програма потребує пристрою, що має можливості геолокації. Уміст елементів типу <uses-feature> використовують для фільтрації списку програм Google Play, доступних для конкретного пристрою.

Елементи <uses-permission> використовують для декларації дозволів на доступ до функцій пристрою, які необхідні застосунку для правильного функціонування. Наприклад, дозвіл android.permission.ACCESS COARSE LOCATION запитують для доступу до функцій визначення місцезнаходження пристрою, а роздільна здатність android.permission.INTERNET для завантаження вмісту з мережі Інтернет. За спроби доступу програми до функцій, які потребують дозволів, не перелічених у файлі маніфесту, випадає виняток. Коли користувач завантажує програму з Google Play, він має переглянути список дозволів і підтвердити доступ програми до функцій, що запитують.

Елемент <application> описує програму загалом. Атрибути android:label та android:icon цього елементу визначають назву застосунку та його іконку, що відображають у списку програм на пристрої. Значенням цих атрибутів можуть бути як текстові рядки, так і посилання на ресурси програми (як у наведеному раніше прикладі). Роботу з ресурсами висвітлено в темі 5.

Усередині елементу <application> розміщують визначення компонентів програми в разі визначення головної активності.

4.3. Складання Android-проєкту

 Складання Android-проєкту можна здійснювати різними способами. У найпростішому випадку використовують утиліту ant, що входить до JDK і є стандартним засобом складання Java-проєктів.

Складальний файл ant зазвичай називають build.xml і розташовують в кореневому каталозі проєкту.

У складальному файлі визначають цілі, кожна з яких відповідає певній операції (наприклад, компіляція, збирання, розгортання, тестування проєкту). Операції складаються з команд, виконання яких приводить до досягнення вказаної мети.

Між цілями може бути встановлено залежності, до того ж команди, необхідні для досягнення певної мети, виконують лише після того, як виконано команди залежних цілей. Крім того, підтримується досить гнучкий

механізм, що дозволяє не виконувати деякі команди, якщо їх уже було виконано раніше та їхнє повторне виконання дасть той самий результат (наприклад, компіляцію модуля не запускають повторно, якщо цей модуль уже було скомпільовано раніше і його вихідний текст не змінювався).

Далі перелічені цілі складального файлу ant, створюваного утилітою Android автоматично:

• help – виводить коротку довідку з усіх автоматично згенерованих цілей;

• debug – складання проєкту в редакції, призначеній для тестування та налагодження програми розробником;

• release – складання проєкту в редакції, призначеній для публікації програми в Google Play (у цьому разі програму після складання буде підписано ключем розробника);

• install – установлення складеного пакета програми в запущеному емуляторі або пристрої. Цю мету можна використовувати тільки разом з однією із цілей складання (debug або release) або вказати суфікс, що означає версію, яку необхідно встановити (installd або installr);

• clean – очищення проєкту, видалення всіх файлів, які містять скомпільований байт-код програми, а також проміжні результати складання.

Слід мати на увазі, що операції, які відповідають будь-якій із зазначених цілей, можна змінювати для того, щоб забезпечити виконання дій, специфічних для конкретного проєкту. Також можна додавати нові цілі.

Для складання проєкту та створення пакета слід виконати команду ant debug.

Якщо додатково потрібно встановити складений пакет у запущений емулятор або на під'єднаний до комп'ютера розробник Android-пристрій, то використовують команду ant debug install.

4.4. Компоненти Android-програми

Перед тим як перейти до розгляду життєвого циклу та можливостей активностей, слід сказати кілька слів про пристрій Android-застосунків, оскільки він суттєво відрізняється від пристрою застосунків як для настільних комп'ютерів, так і для інших мобільних платформ.

 Типовий Android-застосунок складається з компонентів, які можуть належати до таких типів:

• активність (activity) – компонент, що здійснює взаємодію з користувачем;

• сервіс (service) – фоновий процес;

• провайдер контенту (content provider) – компонент, що здійснює надання доступу до даних, що містяться в певному сховищі;

• слухач широкомовних повідомлень (broadcast receiver) – обробник певної глобальної події в операційній системі (наприклад, вимкнення екрана, низький заряд батареї тощо).

Ці компоненти є досить незалежними один від одного та мають чітко визначені інтерфейси для взаємодії з іншими компонентами. Цікава особливість Android-застосунків полягає в тому, що компоненти різних застосунків можуть взаємодіяти між собою. Наприклад, у разі, коли програмі необхідно надіслати повідомлення електронною поштою, воно може викликати стандартну активність із функціоналом поштового клієнта, причому після завершення цієї активності користувач повернеться до роботи з вихідним застосунком. І навпаки, програміст може розробити власний поштовий клієнт і зареєструвати його в системі під час установлення програми, тим самим дозволивши іншим програмам його використання.

Ця особливість дещо розмиває саме поняття застосунку та змушує розглядати всю платформу як відкриту систему, компоненти якої є здатними до кооперативної поведінки. Взаємодія компонентів здійснюється через відправлення асинхронних повідомлень. У застосунку кожне з таких повідомлень асоціюється із сутністю, яку називають "інтент" (intent).

 В Android-**застосунках інтент** – це об'єкт, що інкапсулює в собі запит на виконання певної дії. Інтент може містити такі компоненти:

• дію, яку необхідно виконати (обов'язковий компонент);

• набір категорій, які дають змогу групувати дії;

• URI, що ідентифікує дані, над якими необхідно виконати дію;

• додаткові параметри (extras), необхідні для виконання дії.

Слід зазначити, що інтент, зазвичай, не містить у явному вигляді вказівки адресата повідомлення, що відправляють. Натомість указано дію, яку необхідно виконати. Кожен компонент системи під час установлення реєструє певний набір інтентів, які він може виконувати. Коли

відповідний інтент активовано, система визначає компонент, здатний цю дію виконати, і передає йому управління. Якщо відповідних компонентів є кілька, вибір надають користувачеві.

Кожну активність оголошено у файлі маніфесту.

Фрагмент файлу AndroidManifest.xml, що містить таке визначення, показано на рис. 4.2.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
                    package="ru.ac.uniyar.helloworld"
                android:versionCode="1" android:versionName="1.0">
...
<activity android:name="MainActivity" android:label="@string/app name">
<intent  filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent  filter>
<intent  filter>
<action  android:name="android.intent.action.VIEW"></action>
<data android:scheme="http"></data
category android:name="android.intent.category.DEFAULT" />
</intent  filter>
</activity>
...
</manifest>
```

Рис. 4.2. **Фрагмент файлу AndroidManifest.xml**

Головний елемент оголошення називають <activity>. Параметр android:name цього елементу містить назву активності та його використовують (спільно з назвою кореневого пакета програми) для визначення класу, що містить програмний код цієї активності. Наприклад, у наведеному раніше прикладі такий клас називають ru.ac.uniyar.helloworld.MainActivity, і саме цей клас буде завантажено під час старту активності. Параметр android:label містить назву, яку виводять користувачеві, коли активність відображають на екрані.

Елементи <intent-filter> визначають інтенти, які можуть опрацювати цю активність. У наведеному раніше прикладі визначено два такі інтенти. Перший із них забезпечує можливість відображення програми у списку всіх програм Android та самостійного запуску активності із цього списку.

Для цього вказують тип дії android.intent.action.MAIN та категорію android.intent.category.LAUNCHER. Другий елемент <intent-filter> вказує на те, що активність може функціонувати як стандартний веббраузер, опрацьовуючи дію android.intent.action.VIEW для типу даних http.

Із наведеного прикладу видно, що для визначення дій, які опрацьовує активність, можна використовувати будь-які компоненти інтенту. Докладнішу інформацію про конкретні елементи файлу маніфесту, що дозволяють виконати це визначення, можна знайти в документації API-платформи.

В архітектурі Android-застосунків активності є короткоживучими компонентами. Вони можуть автоматично створюватися та знищуватися в різних ситуаціях, наприклад у разі зміни орієнтації екрана або нестачі пам'яті для інших програм. Крім того, активність може переходити у фон або на передній план, приймати та втрачати фокус. Для правильного опрацювання всіх цих ситуацій уводять поняття життєвого циклу активності та надають засоби, що дозволяють виконувати ті чи ті дії під час переходу між різними фазами цього життєвого циклу.

Життєвий цикл активності містить такі основні стани:

• running/resumed – активність містяться на передньому плані та у фокусі; у цьому стані користувач може безпосередньо взаємодіяти із застосунком за допомогою графічного інтерфейсу;

• paused – активність міститься на передньому плані, але не у фокусі, тобто є перекритою випадним вікном або вікном діалогу; користувач не може безпосередньо взаємодіяти з такою активністю;

• stopped – активність міститься у фоні та не відображається на екрані; користувач не може взаємодіяти з такою активністю.

Переходи між переліченими станами здійснюють за подіями, ініційованими користувачем (наприклад, перемиканням на іншу активність), або системними подіями (наприклад, через брак пам'яті). Кожен перехід супроводжено викликом певних методів у класі Activity, від якого є зобов'язаними успадковувати будь-які активності користувача. Своєю чергою, у класах-спадкоємцях зазначені методи можна перевизначати для того, щоб належним способом відреагувати на перехід між станами життєвого циклу.

Метод onCreate() викликають безпосередньо після створення активності. Тут здійснено ініціалізацію інтерфейсу користувача, прив'язування даних до елементів інтерфейсу, створення потоків і т. ін.

Метод onPause() викликають, коли активність утрачає фокус, у зв'язку з перекриттям її екрана вікном, діалогом або іншою активністю. Після повернення із цього методу активність перейде у стан paused. Зазначмо, що активність, яка перебуває в такому стані, може бути в будь-який момент знищено операційною системою в разі нестачі оперативної пам'яті для інших, можливо більш високопріоритетних застосунків. У зв'язку із цим у методі onPause() необхідно передбачити збереження стану активності, щоб мати можливість відновити його під час перезапуску програми. Відновлення стану здійснюють у callback-методі на Resume ().

На закінчення обговорення життєвого циклу активності звернімо увагу ще одну особливість платформи Android. Зміна системної конфігурації програми, що містить, наприклад, повертання екрана чи зміну локалізації, призводить до знищення активності та її подальшого повторного створення. Водночас виконують усі callback-методи, включно з onDestroy(), для знищуваної та onStart() для новоствореної активності. Для того щоб відрізнити розглянуту ситуацію від ситуації, коли програму закрито користувачем, а потім знову відкрито, передбачено пару методів onSaveInstanceState() та onRestoreInstanceState(). Реалізація цих методів у класі Activity автоматично зберігає та відновлює стан усіх елементів інтерфейсу користувача. Однак у деяких складних випадках може знадобитися перевизначення цих методів для забезпечення коректного функціонування програми в разі зміни конфігурації.

Практична складова до підрозділу 4
Створення застосунку для Android

Мета – опанування студентами умінь створення застосунку для Android.

1. Створення нового проєкту

Запускаймо студію і вибираймо **File | New | New Project** ... З'явиться діалогове вікно майстра.

Поле **Application name:** зрозуміла назва для програми, яка буде відображатися в заголовку програми. За замовчуванням у вас вже може бути **My**Application. Замінімо на Hello.

Поле **Company Domain:** слугує для вказання назви сайта. Уведену назву запам'ятовують і її будуть автоматично підставляти в наступних нових проєктах (рис. 4.3).

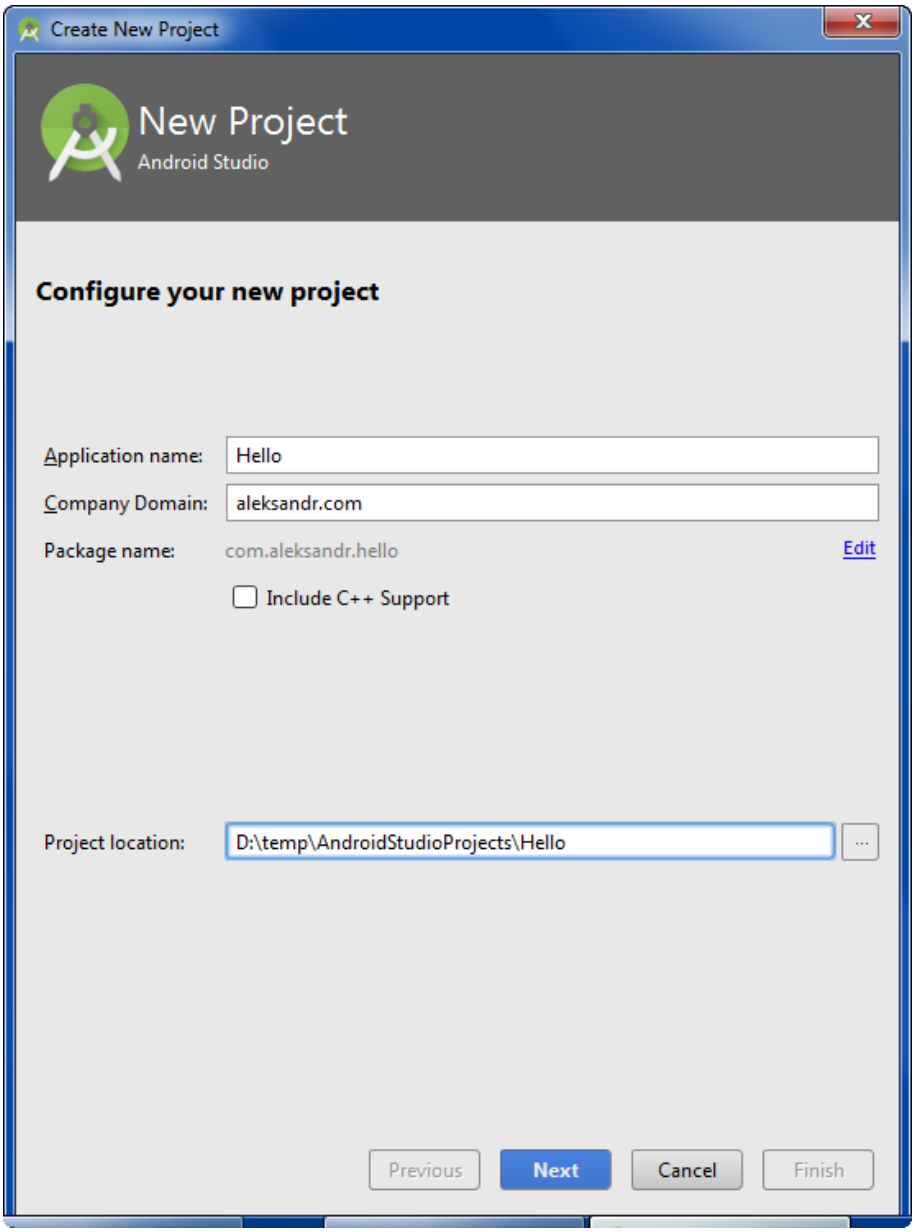


Рис. 4.3. **Створення поля Company Domain**

Поле **Package name:** формує спеціальний Java-пакет на основі вашої назви з попереднього поля. У Java використовують перевернутий варіант для називання пакетів. Пакет слугує для унікальної ідентифікації вашої програми, коли будете його поширювати. Якщо сто осіб напише сто застосунків із назвою Cat, то буде незрозуміло, де застосунок, написаний розробником Василем Котовим. А застосунок із назвою пакета

ru.vaskakotov.cat простіше знайти. Кнопка **Edit** дозволяє відредагувати підготовлений варіант. Наприклад, ви пишете застосунок на замовлення і вам потрібно просто назвати пакунок, затверджений замовником, а не ваш варіант за замовчуванням.

Третє поле **Project location:** дозволяє вибрати місце на диску для створеного проєкту. Виберіть диск D.

Натискаймо на кнопку **Next** і переходьмо до наступного вікна. Тут вибираймо типи пристроїв, під які будемо розробляти свій застосунок. Будемо писати програму для смартфонів і планшетів, тому залишаймо прапорець у першому пункті (рис. 4.4).

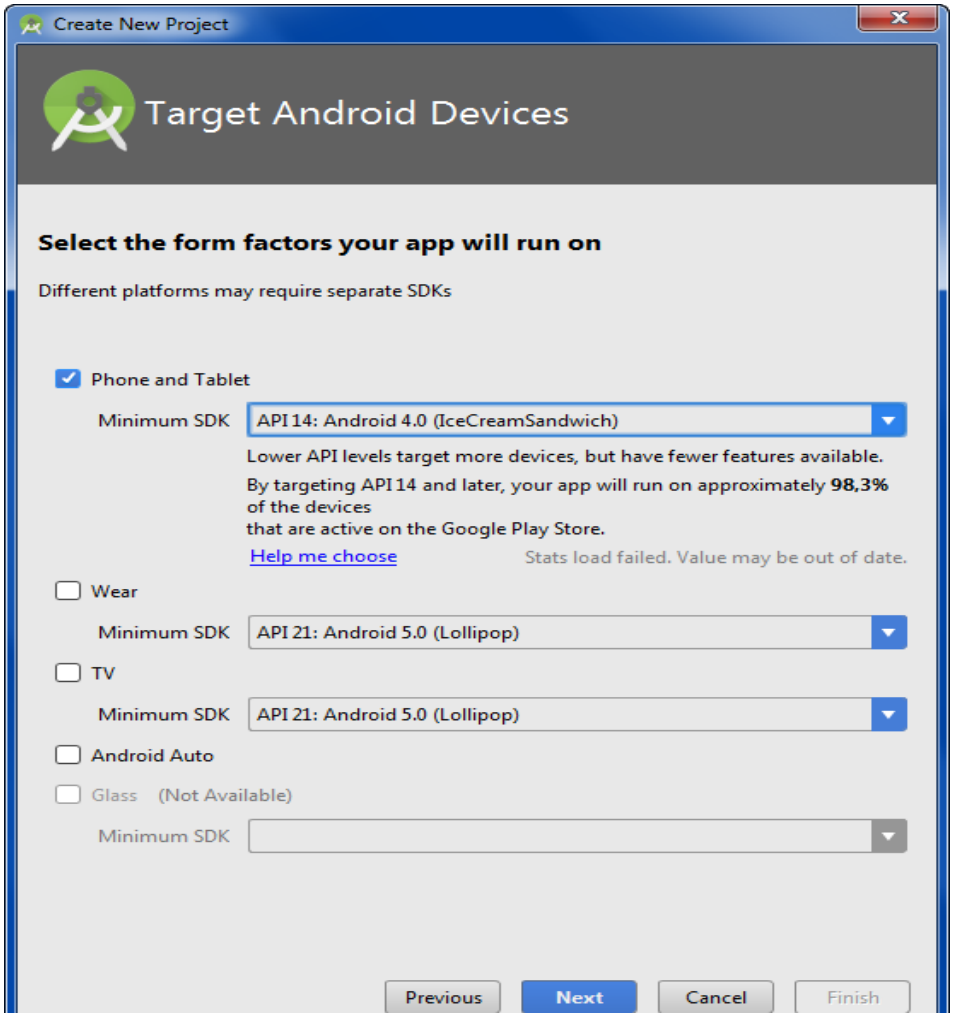


Рис. 4.4. **Вікно вибору типу пристроїв**

Крім вибору типу пристроїв, треба вибрати мінімальну версію системи, під яку буде працювати застосунок. Виберіть Android 4.0. Отже, цю програму може бути запущено на 98 % пристроїв.

Ідімо далі та знову натискаймо Next. Тут слід вибрати зовнішній вигляд екрана програми.

Вибираймо варіант **Empty Activity** та переходьмо до наступного вікна (рис. 4.5).

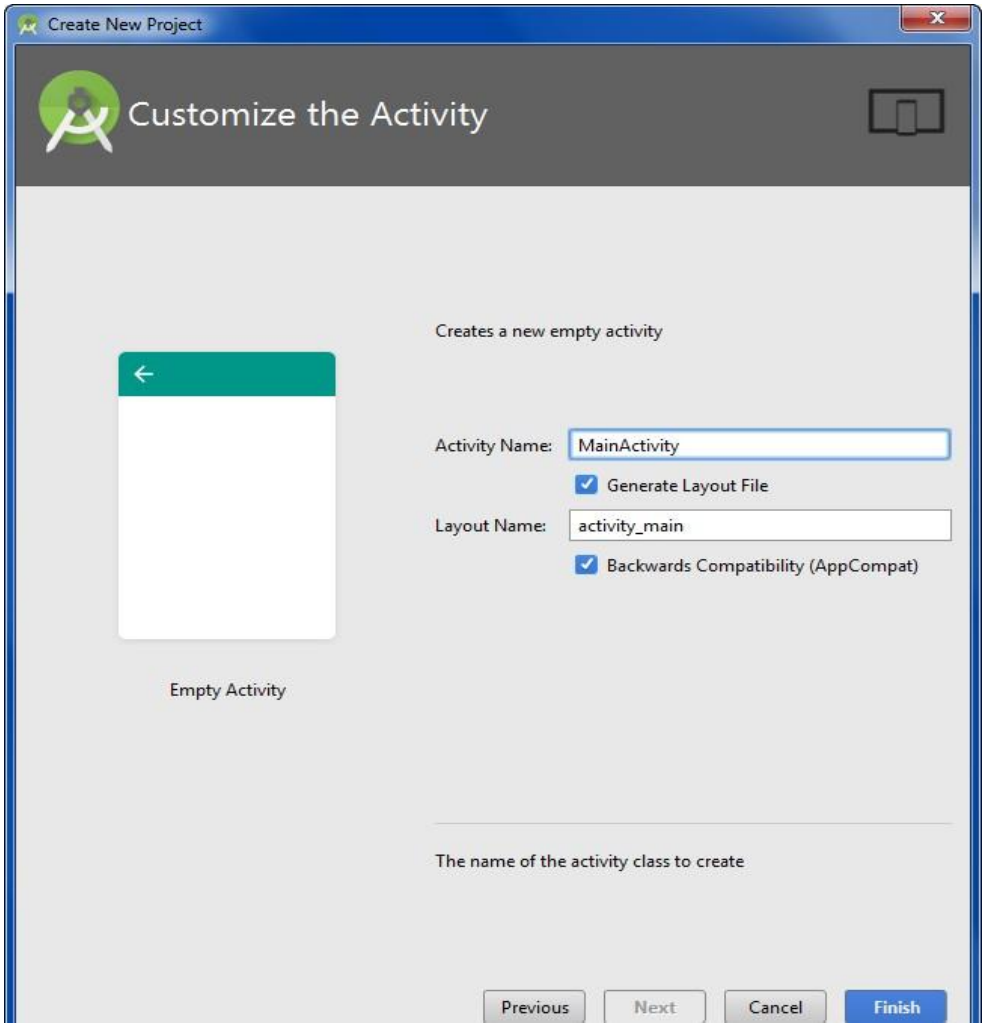


Рис. 4.5. **Вікно вибору варіанта Empty Activity**

На сьогодні закінчується первинне налаштування. Натискаймо кнопку **Finish** і чекаймо. Студія формує проєкт і створює необхідну структуру з різних файлів і папок. Спочатку очі розбігаються. Розберімося.

Бічна ліва частина студії має кілька вертикальних укладок. Імовірніше у вас буде активна перша вкладка **1:** Project. Виберіть **Structure** і **Captures** використовують набагато рідше.

У лівій частині середовища розроблення на вкладці **Android** з'явиться ієрархічний список із папок, які належать до проєкту. У деяких випадках

бажано перемкнутися на режим **Project,** який показує справжнє розташування файлів. Але спершу зручніше використовувати саме вид **Android**, який ховає службові файли.

Зміст проєкту

Укладка **Android** містить дві основні папки: **app** і **Gradle** Scripts.

Перша папка **app** є окремим модулем для програми та містить усі необхідні файли програми – код, ресурси картинок тощо. Друга папка слугує для різних налаштувань, управління проєктом та багатьох інших речей.

Зараз нас має цікавити папка **app**. Розкрийте її. У ній містяться три папки: manifest, java, res.

Папка **manifest** містить єдиний файл маніфесту **AndroidManifest.xml**. У цьому файлі має бути оголошено всі активності, служби, приймачі та контент-провайдери застосунків. Також він має містити необхідні застосунки дозволу. Наприклад, коли програмі слід мати доступ до мережі, це має бути визначено тут. AndroidManifest.xml можна розглядати, як опис для розгортання Android-застосунків.

Папка **java** містить три папки – робочу та для тестів. Робоча папка має назву вашого пакета і містить файли класів. Зараз там один клас **MainActivity**. Папки для тестів можете не чіпати.

Папка **res** містить файли ресурсів, розподілених на окремі папки.

- **Drawable** – у цих папках зберігають графічні ресурси – картинки та xml-файли, що описують колір і фігури.
- **Layout** – у цій папці містяться xml-файли, що описують зовнішній вигляд форм і різних елементів форм. Після створення проєкту там уже є файл **activity_main.xml**, який відповідає за зовнішній вигляд головного вікна програми.
- **Mipmap** – тут зберігають значки застосунків під різні дозволи екрана.
- **Values** – тут розміщують рядкові ресурси, ресурси квітів, стилів і вимірювань, які можна використовувати в цьому проєкті. Тут можна бачити файли colors.xml,dimens.xml,strings.xml, **styles.xml**.

2. Робота із проєктом "Здрастуй, світ!"

Як уже зазначено, програму **Hello, World!** вже вбудовано в будь-який новий проєкт, тому навіть не потрібно нічого писати. Просто потрібно запустити проєкт і здобути готову програму!

Емулятор пристрою краще запустити попередньо через **Tools | Android | AVD** Manager. Емулятор запускають досить тривалий час, особливо якщо процесор комп'ютера не підтримує віртуалізацію.

Не будемо поки вивчати код, а просто натиснімо на зелений трикутник **Run** (Shift + F10) на панелі інструментів у верхній частині студії для запуску програми. Якщо все зробили правильно, то в емуляторі або на пристрої завантажено вашу програму.

Отже, якщо програма запустилася, то побачите вікно програми з написом **Hello World!** (рис. 4.6). Тема у програми збігається з назвою проєкту. Назву проєкту можна знайти у файлі **res/values/strings.xml** і відредагувати, якщо є бажання.

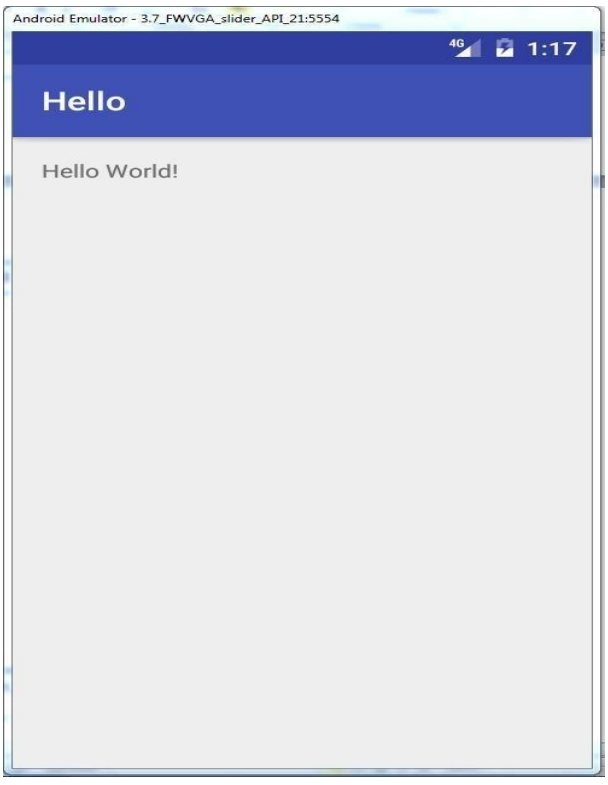


Рис. 4.6. **Вікно програми з написом Hello World!**

Для вивчення потрібно відкрити два файли – **MainActivity** (імовірно його вже відкрито) і **activity_main.xml** (res/layout) у центральній частині

студії. Якщо файли не відкрито, то відкрийте їх самостійно подвійним клацанням для редагування (або перегляду). Таким способом можна відкрити будь-який потрібний файл.

Тепер подивімося на код. Спочатку вивчімо activity_main.xml. Переглядати його можна у двох режимах – **Design** і **Text** (вибір унизу). Відкрийте в режимі Text (рис. 4.7).

```
<?xml version ="1.0" encoding ="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"android:id ="@ + id
    / activity_main" android:layout_width ="match_parent"
    android:layout_height ="match_parent"
    android:paddingBottom ="@ dimen / activity_vertical_margin"
    android:paddingLeft ="@ dimen / activity_horizontal_margin"
    android:paddingRight ="@ dimen / activity_horizontal_margin"
    android:paddingTop ="@ dimen / activity_vertical_margin" tools:context
    ="com.aleksandr.hello.MainActivity">

    <TextView
        android:layout_width ="wrap_content"
        android:layout_height ="wrap_content"
        android:text ="Hello World!" />
</RelativeLayout>
```

Рис. 4.7. **Загальний вигляд activity_main.xml**

Якщо дивитися на монітор, то, замість рядка **android: paddingLeft = "@ dimen / activity_horizontal_margin"**, можна побачити рядок **android: paddingLeft =** "16dp". Однак, якщо підвести курсор мишки до тексту, то побачите підказку. До того ж, якщо ви натиснете на слово, то можна побачити реальний код, який ховається за словами. Це стосується й до інших параметрів, пофарбованих у салатовий прямокутник. Інакше кажучи, студія автоматично вилучає значення з ресурсів і підставляє їх у код, щоб не доводилося згадувати, що ховається за кодом. Комбінація клавіш **Ctrl** +-(мінус) згорне назад.

Щодо XML-коду слід зазначити, що є спеціальний контейнер **RelativeLayout**, у якому розміщено компонент **TextView**, призначений для виведення тексту.

Java-код (MainActivity.java) показано на рис. 4.8.

```
package com.aleksandr.hello;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {@Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState); setContentView
        (R.layout.activity_main);
```

Рис. 4.8. **Java-код (MainActivity.java)**

Назва класу **MainActivity** збігається з назвою файлу з розширенням **java** (це правило, установлене мовою Java). У першому рядку йде назва пакета – її задавали під час створення проєкту (Package Name). Далі йдуть рядки імпорту необхідних класів для проєкту. Для економії місця їх згорнуто в одну групу. Розгорніть її. Якщо одного разу побачите, що назви класів виведено сірим кольором, значить їх не використовують у проєкті (підказка **Unused import** statement) і можна спокійно видалити зайві рядки.

Далі йде оголошення самого класу, який успадковано (extends) від абстрактного класу **Activity**. Це базовий клас для всіх екранів застосунку. Не виключено, що у вас буде **AppCompatActivity**, якщо під час створення проєкту ви залишили підтримання старих пристроїв (прапорець **Backwards Compatibility (AppCompat))**. Клас **AppCompatActivity** якраз і належить до бібліотеки сумісності. Уважайте її бідним родичем базової **Activity**. У неї є всі потрібні методи й допоміжні класи, але назви можуть трохи відрізнятися. І змішувати їх не можна. Якщо вже використовуєте клас із бібліотеки сумісності, то методи беріть відповідні.

У самому класі бачимо метод **onCreate ()** – він викликається, коли застосунок створює та відображає розмітку активності. Метод позначено як **protected** і супроводжено анотацією @**Override** (перевизначений із базового класу). Анотація може стати у пригоді. Якщо зробили помилку в назві методу, то компілятор зможе попередити про це, повідомивши про відсутність такого методу у класу Activity.

Розберімо код методу.

Рядок **super.onCreate (savedInstanceState);** – це конструктор батьківського класу, що виконує необхідні операції для роботи активності. Цей рядок вам не доведеться чіпати, залишайте без змін.

Другий рядок **setContentView (R.layout.activity_main);** становить більший інтерес. Метод **setContentView (int)** підключає вміст із файлу розмітки. Як аргумент указують назву файлу без розширення з папки **res/layout**. За замовчуванням проєкт створює в ньому файл activity_main.xml.

Щоб ваш код був акуратним, намагайтеся дотримуватися стандартів. Якщо створюєте розмітку для активності, то використовуйте префікс **activity** для назви файлу. Наприклад, розмітка для другої активності може мати назву activity_second.xml.

Отже, створено нову програму, але це ще не привід уважати себе програмістом, оскільки ви не написали жодного рядка коду. Настав час набратися сміливості та створити програму **Hello Friend!**. Нині ця програма є занадто простою. Уявіть собі, що у вас на екрані має бути розташовано кілька кнопок, текстових полів, картинок. Кожному об'єктові потрібно задати розміри, координати, колір, текст і т. ін. Android підтримує спосіб, заснований на XML-розмітці, який буде нагадувати розмітку вебсторінки. Початківці програмісти можуть використовувати візуальний спосіб перетягування об'єктів за допомогою мишки. Більш просунуті можуть писати код вручну. Частіше використовують комбінований підхід.

Файли XML-розмітки містяться в папці **res/layout** цього проєкту. Слово res є скороченням від слова resources (ресурси). Папка містить ресурси, які не пов'язано з кодом. Крім розмітки, там же містяться зображення, звуки, рядки для локалізації тощо.

Розкрийте зліва у структурі проєктів папки **res/layout** і двічі клацніть на файлі **activity_main.xml**, якщо він у вас є закритим. Зверніть увагу, що XML-файли можна переглядати у двох режимах: текстовому та візуальному. Для цього призначено дві вкладки в нижній частині вікна редактора: **DesigniText** (рис. 4.9).

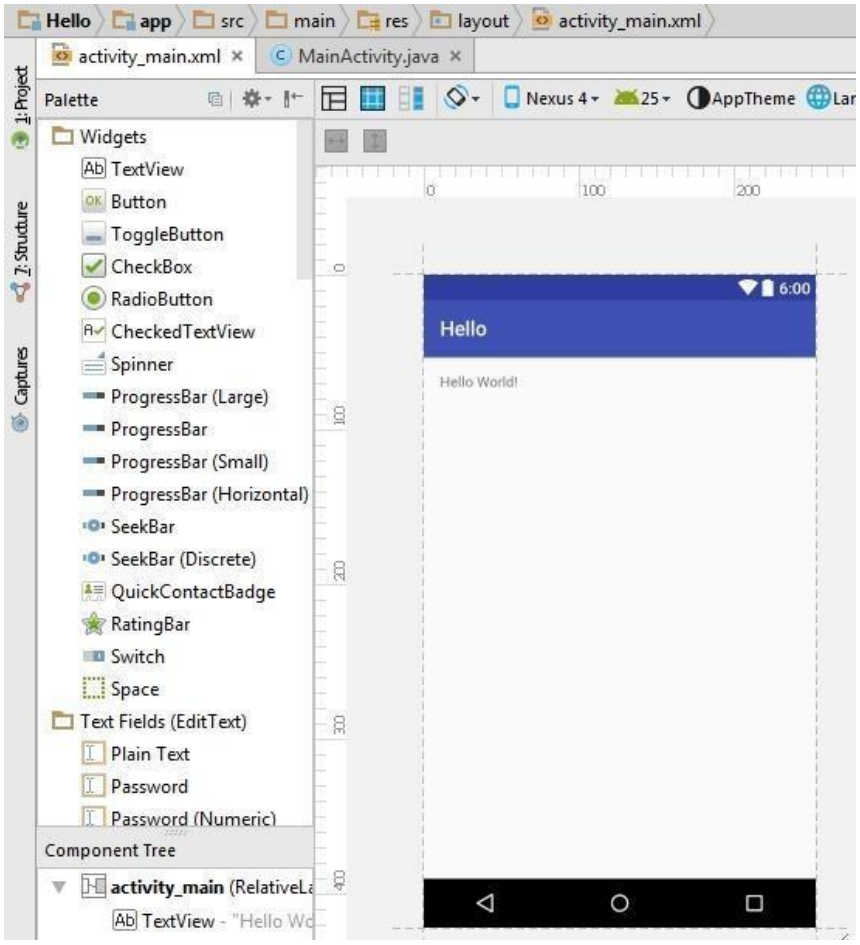


Рис. 4.9. **Вікно редактора DesigniText**

Коли розмітку відкрито у графічному поданні, то зліва від основної частини редактора коду можна побачити панель інструментів, у якому зібрано різні елементи по групах **Widgets, TextFields, Layouts** і т. ін. У групі **Images & Media** знайдіть елемент **ImageButton**, перетягніть його на форму і відпустіть. Точне розташування нас не цікавить, але постарайтеся розмістити компонент у центрі екрана активності. У вас з'явиться діалогове вікно із проханням вибрати зображення для кнопки. Поки вибирайте будь-яку, наприклад, першу. Потім замініть.

Тепер навчімося змінювати фон для екрана програми. Зараз у нас екран білого кольору. Підготуймо колірний ресурс із потрібним кольором.

Повертаймося у файл розмітки **activity_main.xml**. Справа знайдіть укладку **Properties**, у якій відображено властивості для вибраного елементу. Новачки часто плутаються спершу й починають змінювати властивості не в тих елементів, які їм були потрібними. Зараз у вас є сама форма, графічна кнопка **ImageButton** і текстова мітка **TextView** із написом **Hello World!**. Поклацайте по цих елементах, щоб побачити, як змінюється

зміст властивостей у панелі властивостей. Оскільки збираємося працювати з фоном екрана програми, то клацніть у зоні форми (можна також вибрати через Component Tree внизу зліва). У панелі "Властивість" відобразяться найуживаніші властивості вибраного компонента. До них належать ідентифікатор, ширина та висота (рис. 4.10).

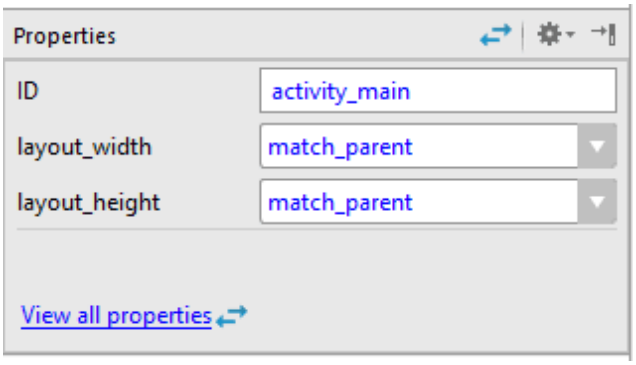


Рис. 4.10. **Налаштування ідентифікатора, ширини та висоти**

Клацаймо на посилання **View allproperties**, щоб відкрити всі властивості компонента.

Знайдіть властивість **background**. Клацніть поруч із цим словом у другій колонці, де потрібно прописувати значення. З'явиться текстове поле, у яке можна ввести значення вручну, і кнопка із трьома крапками, яка запустить діалогове вікно для створення ресурсу (рис. 4.11).

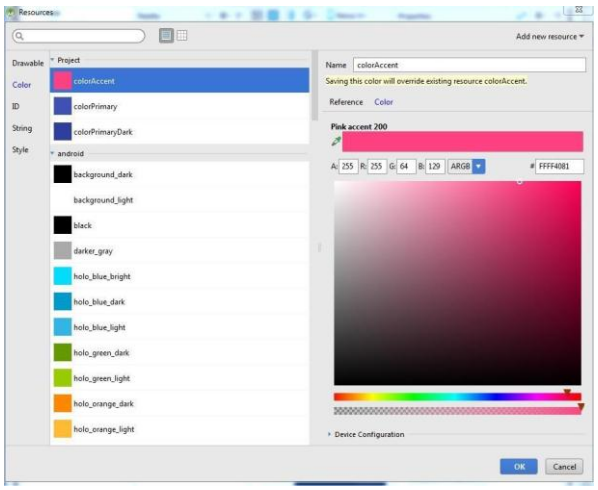


Рис. 4.11. **Діалогове вікно для створення ресурсу**

Переходьмо на вкладку **Color** і вибираймо колір. Зверху в розділі **Project** показано кольори, визначені у проєкті у стилі Material Design.

У розділі **Android** показано кольори, наявні в ресурсах системи. За бажання можна встановити свій колір. Виберемо колір **colorAccent** і натиснімо кнопку OK.

Екран забарвлюється в рожевий колір.

Якщо переключитися в текстовий режим, то побачимо, що в елементу **RelativeLayout** додався рядок (рис. 4.12).

```
android:background ="@ color / colorAccent"
```

Рис. 4.12. **Додатковий рядок RelativeLayout**

Далі поміняймо картинку для графічної кнопки (рис. 4.13). Знаходьмо відповідне зображення й копіюймо його в папку **res/drawable**.

Рис. 4.13. **Картинка для графічної кнопки**

Просте перетягування із провідника в папку студії не спрацює. Тому краще скопіювати картинку в буфер, потім натиснути правою кнопкою мишки на папці **drawable** у студії, вибрати команду "Вставити" (рис. 4.14).

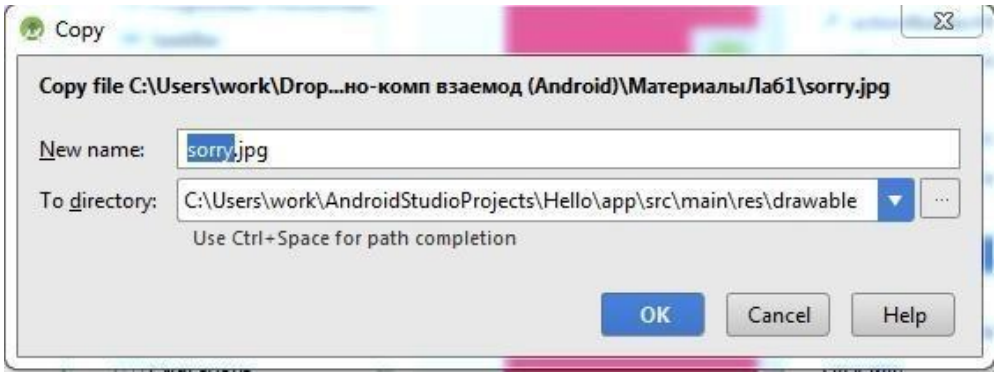


Рис. 4.14. **Команда "Вставити"**

Коли помістити графічний файл у зазначену папку, то студія автоматично створює ресурс типу **Drawable** із назвою файлу без розширення, до якого можна звертатися програмно.

Виділяймо елемент **ImageButton** на формі та в панелі властивостей відкриймо тільки важливі властивості, вибираймо властивість **srcCompat**. Знову клацаймо на кнопці із трьома крапками та вибираймо ресурс в категорії **Drawable** – там можна побачити ресурс *sorry* (назву доданого раніше файлу, рис. 4.15).

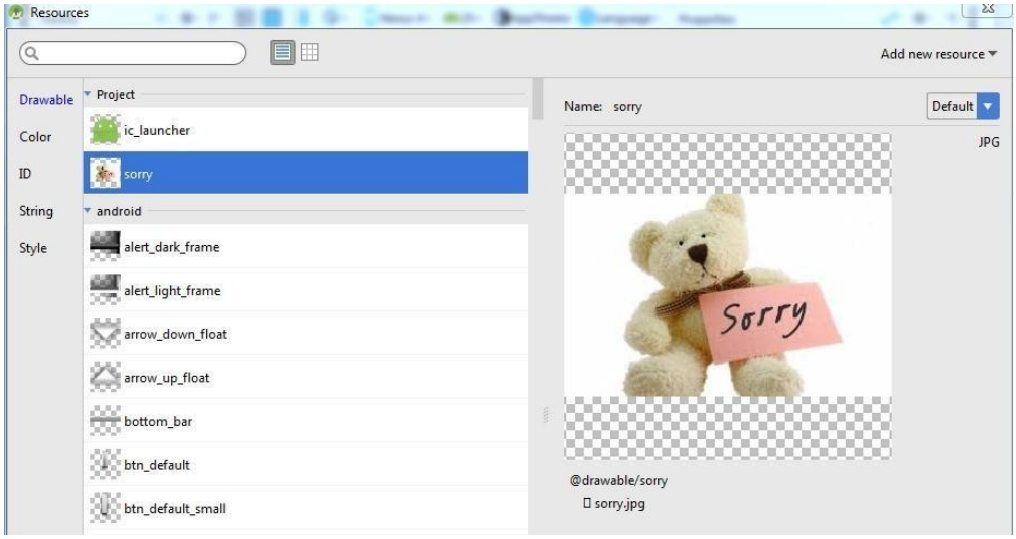


Рис. 4.15. **Вибір ресурсу в категорії Drawable**

Запам'ятайте, що назви ресурсів мають починатися з літери й можуть містити літери та цифри, а також знак нижнього підкреслення. Інші символи (типу тире, решітки тощо) використовувати не можна.

У вікні властивостей слід знайти властивість **onClick** і вручну написати **onClick** – це назва методу для опрацювання натискання на кнопку. Можна придумати й іншу назву, наприклад, onButtonPressed.

Закінчімо роботу із графічним інтерфейсом програми. Наостанок, слід виділити елемент **TextView** із написом **Hello World** і видалили його. У розділі **Widgets** знайдіть компонент **TextView** і перетягніть його на форму застосунку. Постарайтеся розмістити його під графічної кнопкою з малюнком.

У цього компонента буде щось написано у властивості **ID**. Імовірніше, це буде **textView 2**. Запам'ятайте його.

Текст дрібнуватий, тому у властивості **textAppearance** встановіть значення AppCompat. Display2 (рис. 4.16).

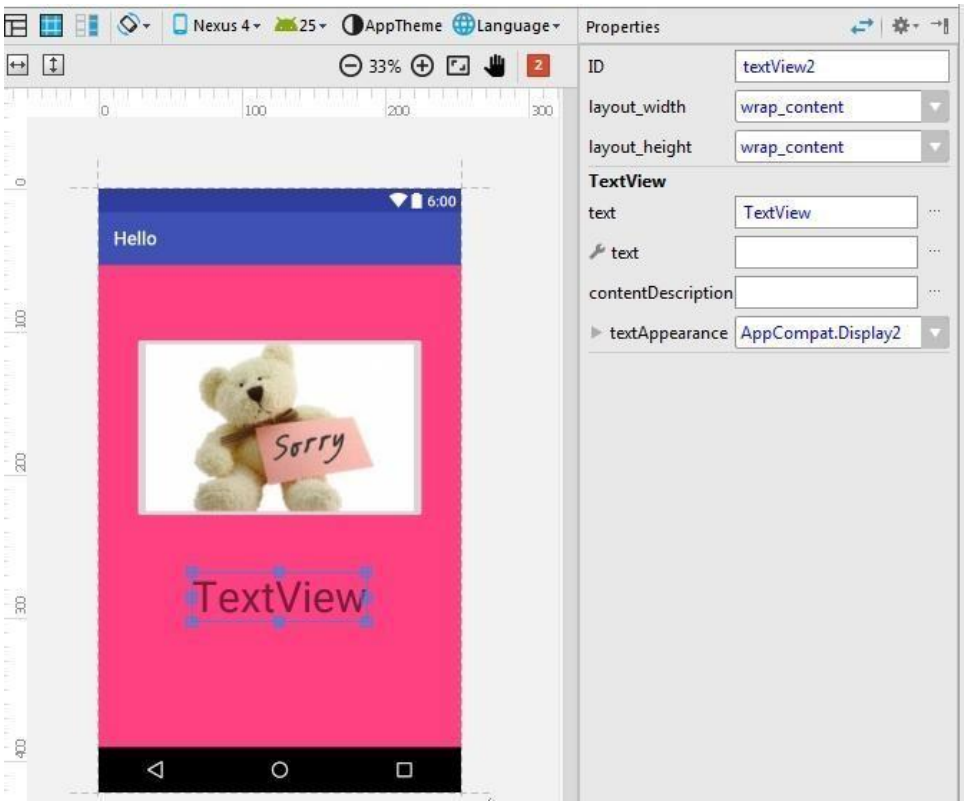


Рис. 4.16. **Коригування властивості textAppearance**

Вийшов такий код (рис. 4.17).

```
<?xml version ="1.0" encoding ="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id ="@ + id / activity_main" android:layout_width ="match_parent"
    android:layout_height ="match_parent"
    android:paddingBottom ="@ dimen / activity_vertical_margin" android:paddingLeft
    ="@ dimen / activity_horizontal_margin"
    android:paddingRight ="@ dimen / activity_horizontal_margin"

    android:paddingTop ="@ dimen / activity_vertical_margin" tools:context
    ="com.aleksandr.hello.MainActivity" android:background ="@ color /colorAccent">
    <ImageButton
        android:layout_width ="wrap_content" android:layout_height ="wrap_content"
        app:srcCompat ="@ drawable / sorry" android:layout_marginTop ="58dp"
        android:id ="@ + id / imageButton" android:onClick ="onClick"
        android:layout_alignParentTop ="true" android:layout_centerHorizontal ="true"
        />
```

Рис. 4.17. **Код коригування властивості textAppearance**

```
    <TextView
        android:text ="TextView" android:layout_width ="wrap_content"
        android:layout_height ="wrap_content"
        android:layout_below ="@ + id / imageButton"
        android:layout_centerHorizontal ="true"
        android:layout_marginTop ="55dp"
        android:id ="@ + id / textView2"
        android:textAppearance ="@ style / TextAppearance.AppCompat.Display2"
/>
</RelativeLayout>
```

Закінчення рис. 4.17

Установіть курсор мишки всередині тексту **"onClick"** біля кнопки та натисніть комбінацію **Alt+Enter**. У випадному вікні виберіть варіант **Create 'onClick (View)' in MainActivity'** (рис. 4.18).

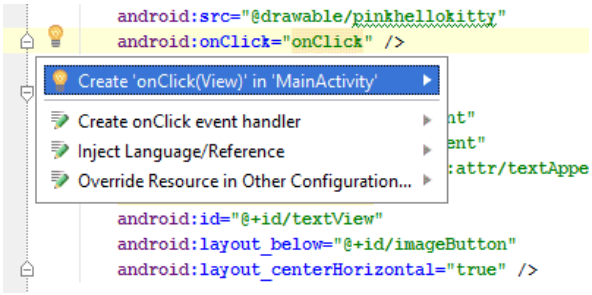


Рис. 4.18. **Варіант Create 'onClick (View)' in 'MainActivity'**

У коді класу **MainActivity** з'явиться заготовка для опрацювання клацання кнопки.

Раз уже в нас тепер відкрито файл **MainActivity.java**, то продовжмо тепер роботу в ньому. Оскільки збираймося змінювати текст у текстовій мітці, необхідно прописати цей елемент у коді. До методу **onCreate ()** наберіть такий рядок (рис. 4.19).

```
private TextView mHelloTextView;
```

Рис. 4.19. **Додатковий рядок до методу onCreate ()**

Оголосімо змінну типу **TextView** під назвою mHelloTextView.

Якщо набирали вручну й за підказками використовували клавішу Enter, то клас **TextView** автоматично імпортується та запис з'явиться в секції **import**. Якщо просто копіюєте текст із сайта, то студія підкреслить назву класу **TextView** і запропонує імпортувати його вручну.

Далі всередині методу **onCreate ()** після виклику методу **setContentView ()** додайте рядок (рис. 4.20).

```
mHelloTextView = (TextView) findViewById (R.id.TextView2); //пам'ятайте,
я просив запам'ятати ідентифікатор
```

Рис. 4.20 **Додатковий рядок до методу setContentView ()**

Уникайте спокуси скопіювати рядок із методички та вставити в код, пишіть самостійно й активно використовуйте автозавершення (Enter) під час набору слів. Студія часто сама активно допомагає підказками. Тепер система знає про наявність елементу **TextView**, і можна до нього звертатися для зміни різних властивостей, наприклад, змінити текст.

Переходьмо до заготовки для клацання кнопки (рис. 4.21).

```
public void onClick(View view) {
}
```

Рис. 4.21. **Заготовка для клацання кнопки**

Далі просто пишімо код між фігурними дужками (рис. 4.22).

```
mHelloTextView.setText("Hello Friend!")
```

Рис. 4.22. **Код між фігурними дужками**

Звернімося до елементу **mHelloTextView** і через його метод **setText()** програмно змінімо текст на потрібні слова.

Запускаймо програму й натискаймо на кнопку із зображенням малюнка. Якщо все зроблено правильно, то відобразиться чудова фраза.

Із цієї миті можна вважати себе справжнім програмістом – ви навчилися створювати колірні та графічні ресурси, змінити фон у **застосунку** через XML-розмітку, опрацьовувати натиснення кнопки та виводити текстові повідомлення.

У папці **appbuildoutputsapk** проєкту можна знайти готовий APK-файл, який можна дати скачати знайомим (у телефоні має бути дозвіл на встановлення непідписаних застосунків).

Кожному застосунку виділяємо певний обсяг пам'яті під картинки. У нових пристроях трохи більше, у старих – поменше. Але у будь-якому разі не треба намагатися завантажити в цьому прикладі фотографію свого улюбленого авто обсягом декілька десятків мегабайтів, інакше можете дістати помилку в застосунку.

Контрольні запитання для самоперевірки

1. Що таке Android SDK? Які компоненти містить? Які інструментальні засоби можна використовувати під час розроблення програм на платформі Android?

2. Що таке "менеджер пакетів Android"? Які завдання він вирішує?

3. Яка структура проєкту, що автоматично створюється, для Android? Які компоненти створюють та у яких каталогах їх розміщують?

4. Що таке "файл маніфесту"? Яка його структура? Які основні елементи можна зустрічати у файлі маніфесту та для чого вони є потрібними?

5. Що таке ant? Як його використовують для збирання програм? Які цілі містяться в автоматично згенерованому файлі складання?

Рекомендована література: [2; 4; 5 – 9; 11; 13; 15 – 18].

5. Робота з ресурсами в операційній системі Android

Мета – ознайомлення студентів зі специфікою роботи з ресурсами в Android.

Професійні компетентності: здатність застосовувати сучасні методи й інструменти для досліджень у сфері видавництва та поліграфії, а також забезпечення якості продукції; здатність аналізувати структури та контент проєктів інтерактивних медіа.

5.1. Поняття ресурсів, їхня класифікація та призначення

 **Ресурси в Android** – це статичні дані (наприклад, текст, зображення, опис інтерфейсу користувача), що є частиною програми. Ресурси розміщує розробник усередині проєкту у вигляді файлів і їх переносять в apk-пакет програми автоматично під час складання.

Використання ресурсів має дві основні цілі:

1. Відділення даних від коду. Під час використання ресурсів дані зберігають окремо від коду в декларативному вигляді, тому їх легко змінювати, причому зміни можуть уносити не програмісти, а, наприклад, дизайнери інтерфейсу користувача або перекладачі. Зміна ресурсів не потребує перекомпіляції програми – необхідно лише її перескладання.

2. Забезпечення варіативності використовуваних ресурсів, залежно від конфігурації. Ресурси дозволяють реалізувати підтримання різних типів орієнтації екрана та різних мов інтерфейсу користувача без додаткових витрат із боку програміста, оскільки ресурси, які підходять для поточної конфігурації системи, завантажують автоматично.

 В Android виділяють такі основні типи ресурсів.

• Layout – файли у форматі XML, які описують розташування елементів інтерфейсу користувача. Приклад такого файлу розглядали раніше в п. 3.3.

• Menu – файли у форматі XML, які описують компонування елементів меню або панелі дій. Використання цих файлів буде висвітлено далі в п. 5.5.

• Drawable – файли зображень, що використовує програма. Сюди також входять графічні файли, які використовують в інтерфейсі користувача.

• Values – дані програми, подані в текстовому форматі XML.

Ресурс у програмі Android становить собою файл, наприклад, файл розмітки інтерфейсу, або деяке значення, наприклад, простий рядок. Тобто ресурсами є і файли розмітки, й окремі рядки, і звукові файли, файли зображень і т. ін. Усі ресурси містяться у проєкті в каталозі res. Для різних типів ресурсів, визначених у проєкті, у каталозі res створюють підкаталоги. Підкаталогами, що підтримують, слугують такі:

animator/: xml-файли, що визначають анімацію властивостей;

anim/: xml-файли, що визначають tween-анімацію;

color/: xml-файли, які визначають список кольорів;

drawable/: графічні файли (.png, .jpg, .gif);

mipmap/: графічні файли, що використовують для іконок програми під різні роздільні здатності екранів;

layout/: xml-файли, що визначають інтерфейс користувача програми;

menu/: xml-файли, що визначають меню програми;

raw/: різні файли, які зберігають у вихідному вигляді;

values/: xml-файли, які містять різні значення, що використовують у застосунку, наприклад, ресурси рядків;

xml/: довільні xml-файли;

font/: файли з визначеним шрифтом та розширеннями .ttf, .otf або .ttc, або файли XML, які містять елемент <font-family>.

Загалом можна визначити такі типи ресурсів (табл. 5.1).

Таблиця 5.1

Типи ресурсів в Android

Ресурси	Каталог проєкту	Файли	Елемент у файлі
1	2	3	4
Рядки	/res/values/	strings.xml	<string>
Plurals	/res/values/	strings.xml	<plurals>
Масиви рядків	/res/values/	strings.xml або arrays.xml	<string-array>
Логічні значення Boolean	/res/values/	bools.xml	<bool>
Кольори	/res/values/	colors.xml	<color>
Список кольорів	/res/color/	Вільна назва	<selector>
Розміри (Dimensions)	/res/values/	dimens.xml	<dimen>
Ідентифікатори ID	/res/values/	ids.xml	<item>
Цілі числа	/res/values/	integers.xml	<integer>
Масив цілих чисел	/res/values/	integers.xml	<integer-array>
Графічні файли	/res/drawable/	Файли з роздільними здатностями jpg і png	–
Tween-анімація	/res/anim/	Файл xml із будь-якою назвою	<set>, <alpha>, <rotate>, <scale>, <translate>
Покадрова анімація	/res/drawable/	Файл xml із будь-якою назвою	<animation-list>

Закінчення табл. 5.1

1	2	3	4
Анімація властивостей	/res/animator/	Файл xml із будь-якою назвою	<set>, <objectAnimator>, <valueAnimator>
Меню	/res/menu/	Файл xml із будь-якою назвою	<menu>
XML-файли	/res/xml/	Файл xml із будь-якою назвою	
Бінарні та текстові ресурси	/res/raw/	Файли мультимедіа (mp3, mp4), текстові та інші файли	
Розмітка графічного інтерфейсу	/res/layout/	Файл xml із будь-якою назвою	
Стилі та теми	/res/values/	styles.xml, themes.xml	<style>

Наприклад, якщо візьмемо стандартний проєкт Android Studio, який створюють за замовчуванням, то там можна помітити наявність кількох папок для різних ресурсів у каталозі res, який показано на рис. 5.1.

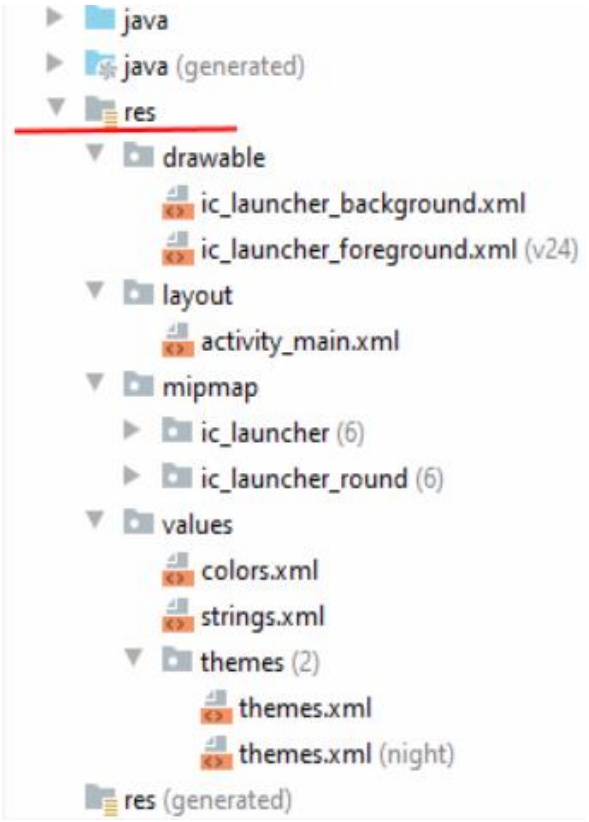


Рис. 5.1. **Каталог res стандартного проєкту Android Studio**

За замовчуванням тут є каталоги не для всіх типів ресурсів, які використовують в Android, проте за необхідності можна додати в папку res потрібний каталог, а потім помістити ресурс. Raw – довільні дані, зазвичай, бінарні.

Коли відбувається компіляція проєкту, відомості про всі ресурси додають у спеціальний файл R.jar, який потім використовують під час роботи з ресурсами.

 Насамперед до складу ресурсів цього типу входять усі текстові рядки програми, які традиційно розміщують у файлі strings.xml. Приклад такого файлу показано на рис. 5.2.

```
<?xml version="1.0" encoding="utf 8"?>
<resources>
<string name="app name">Alarm clock</string>
<string name="hours label">Hours</string>
<string name="minutes label">Minutes</string>
</resources>
```

Рис. 5.2. **Приклад файлу strings.xml**

Ресурси кожного типу розміщують усередині каталога res проєкту в підкаталогах, назви яких збігаються з типами ресурсів, наведеними раніше, але написаними малими літерами.

Для використання ресурсів із програми під час складання проєкту (під час використання інструментальних середовищ також за будь-якої зміни ресурсів програми) створюють спеціальний файл R.java, що містить ідентифікатори всіх ресурсів проєкту. Усі їх визначено як статичні константи всередині підкласів класу R, кожен із яких відповідає своєму типу ресурсів:

• R.layout – ресурси з каталога res/layout, що описують компонування інтерфейсу користувача;

• R.menu – ресурси з каталога res/menu, які описують склад меню чи панелі дій;

• R.id – компоненти інтерфейсу користувача, описані у файлах ресурсів (файли каталога res/layout); елементи розрізняють за ідентифікаторами, що задають за допомогою атрибута android:id;

• R.drawable – ресурси-зображення з каталога res/drawables;

• R.string, R.integer, R.boolean, R.color, R.array тощо – ресурси з файлів даних (файли каталога res/values), згруповані за типами цих даних;

• R.raw – інші ресурси каталога res/raw.

Наведімо приклади використання ресурсів із коду. Завантаження ресурсу, що описує інтерфейс програми, із файлу res/layout/main.xml здійснюють за допомогою виклику:

setContentView(R.layout.main);

Читання текстового ресурсу, визначеного у файлі res/strings.xml як

```
<?xml version="1.0" encoding="utf 8"?>
<resources>
<string name="error message">Error!</string>
</resources>
```

Виведення інформації, що міститься в цьому ресурсі, у вигляді випадного повідомлення можна виконати таким способом:

```
Toast.makeText(this, R.string.error message, Toast.LENGTH LONG).show();
```

Наведені раніше приклади використовували лише ідентифікатори ресурсів, які передавали різним віджетам, а останні, своєю чергою, завантажували необхідні ресурси. Але є можливість безпосереднього завантаження за допомогою методів класу Resources. Об'єкт цього класу можна дістати через виклик getResources() класу Context:

```
Resources resources = getResources();
```

Потім можна дістати текстовий ресурс і ресурс-зображення таким способом:

```
String text = resources.getText(R.string.app name);
Drawable icon = resources.getDrawable(R.drawable.ic launcher);
```

Також є можливість посилання ресурсів з опису інших ресурсів. Її використовують для визначення ідентифікаторів елементів у файлах опису інтерфейсу користувача:

```
<TextView android:id="@+id/counterTextView" />
```

підстановки рядків із файлів res/values як написи віджетів:

```
<TextView android:text="@string/counterText" />
```

а також у файлі маніфесту:

```
<application android:label="@string/app name"
android:icon="@drawable/ic launcher">
```

В останньому випадку вказують текстовий ресурс, що містить назву програми, та ресурс-зображення, що є його іконкою.

5.2. Ресурси конфігурації

Як уже згадувалося, однією із цілей використання ресурсів є підтримання їхньої варіативності, залежно від конфігурації системи. Найбільш важливими елементами конфігурації, що потребують такої варіативності, є:

• роздільна здатність екрана (у програмі необхідно використовувати зображення, що відповідають для встановленої роздільної здатності екрана користувача, щоб уникнути їхньої якості під час масштабування);

• орієнтація екрана (для книжкової та альбомної орієнтації, зазвичай, потрібне різне компонування елементів інтерфейсу користувача);

• поточна локаль (інтерфейс користувача програми має бути тією мовою, яку встановлено в системі).

Для вирішення завдання варіативності файли ресурсів, що відповідають різним конфігураціям, розміщують у різних каталогах. До того ж суфікси вказують, на яку конфігурацію призначено ресурси з відповідного каталога:

• layout – каталог опису інтерфейсу для книжкової орієнтації; layout-land – для альбомної;

• drawable – набір стандартних зображень; drawable-ldpi, drawable-mdpi, drawable-hdpi – набори зображень для пристроїв із низькою, середньою та високою роздільною здатністю, відповідно;

• values – рядки програми для мови за замовчуванням (англійською), values-ua – рядки програми в українській локалізації, values-de – рядки програми в німецькій локалізації тощо. Наведімо приклад використання цього механізму для інтернаціоналізації програми. Нехай файл ресурсів, наведений у підрозд. 5.1, має назву strings.xml і міститься всередині каталога res/values проєкту. Тоді для того щоб забезпечити інтернаціоналізацію програми для української мови, необхідно створити такий файл та розмістити його під назвою strings.xml усередині каталога res/values-ua (рис. 5.3).

```
<?xml version="1.0" encoding="utf 8"?>
<resources>
<string name="app name">Будильник</string>
<string name="hours label">Годинник</string>
<string name="minutes label">Хвилини</string>
</resources>
```

Рис. 5.3. **Файл для розміщення всередині каталога res/values-ua**

Розгляньмо ще один приклад використання ресурсів, який демонструє формування головного меню. У ранніх версіях Android головне меню програми викликають натисканням на апаратну кнопку Menu. Починаючи з Android 3, використовують інший підхід: елементи меню вишиковують у правому кутку рядка заголовка програми, яку в цьому разі називають *панеллю дій (action bar).* Принцип реалізації є однаковим для обох механізмів. Наведімо його опис.

Спочатку меню потрібно описати у файлі ресурсів. У цьому разі будемо вважати, що такий файл має назву res/menu/main.xml (рис. 5.4).

```
<?xml version="1.0" encoding="utf  8"?>
<menu  xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/menu save" android:icon="@drawable/ic menu save"
android:title="@string/menu save" android:showAsAction="ifRoom withText" />
item android:id="@+id/menu delete" android:icon="@drawable/ic menu delete"
android:title="@string/menu delete" android:showAsAction="ifRoom withText" />
<item android:id="@+id/menu options" android:title="@string/menu options"
android:showAsAction="never" />
</menu>
```

Рис. 5.4. **Файл res/menu/main.xml**

До складу меню входять три пункти. Перший і другий оформлено як елементи панелі дій (атрибут android:showAsAction є і не одного never), для кожного з них визначено значок і текст (атрибути android:icon та android:title, відповідно). Значення ifRoom|withText означає, що в панелі дій мають відображати як значок дії, так і її текстове опис (останнє лише за наявності достатньої кількості вільного простору в панелі). Інші можливі значення містять always, що означає, що елемент завжди мають

відображати, і never, що означає, що елемент не буде відображено в панелі дій. Це використовують в описі третього елементу в наведеному раніше прикладі. Користувач зможе здобути доступ до цього елементу лише шляхом натискання кнопки Menu. Скриншоти програми з відкритим меню в портретній та альбомній орієнтації показано на рис. 5.5.

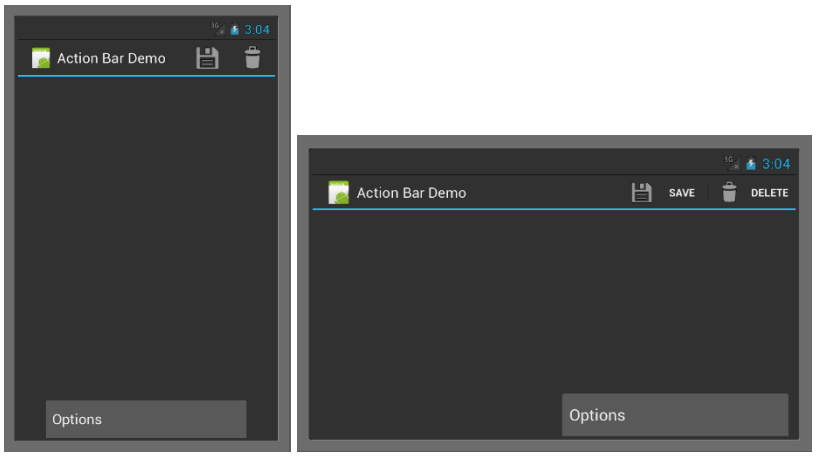


Рис. 5.5. **Панель дій у портретній та альбомній орієнтації**

Щоб завантажити наведений опис у програмному коді, необхідно перевизначити метод onCreateOptionsMenu() класу активності (рис. 5.6).

```
@Override
public boolean onCreateOptionsMenu(Menu menu)MenuInflater inflater =
getMenuInflater(); inflater.inflate(R.menu.main, menu);
return true;
}
```

Рис. 5.6. **Перевизначення методу onCreateOptionsMenu()**

Цей метод передає параметр menu, який необхідно використовувати для формування меню. У наведеному коді цю операцію здійснено за допомогою класу MenuInflater, який здатний завантажувати меню із XML-файлів опису. Метод inflate() передає ідентифікатор ресурсу, із якого необхідно завантажити опис, а також об'єкт-меню, що підлягає формуванню.

Для опрацювання дій меню та панелі завдань необхідно перевизначити метод onOptionsItemSelected() класу активності (рис. 5.7).

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
swit (item.getItemId())
case R.id.menu save:
// Handle the "save" operation
break;
case R.id.menu delete:
// Handle the "delete" operation
break;
// …
}
return true;
}
```

Рис. 5.7. **Перевизначення методу onOptionsItemSelected()**

Цей метод є диспетчером, тому типовий спосіб його реалізації полягає у визначенні дії, яку було вибрано, із наступним опрацюванням цієї дії.

5.3. Виклик активності через інтент


Виклик активності здійснюють через інтент. У програмному коді для цього необхідно створити екземпляр класу Intent, а потім передати цей екземпляр методу startActivity(), визначеному у класі Context – супер-класі Activity. Наприклад:

```
Intent intent = new Intent(Intent.ACTION VIEW,
    Uri.parse("http://developer.android.com"));
startActivity(intent);
```

Тут створено інтент зі стандартною дією ACTION VIEW (цю дію асоційовано з відкриттям інтернет-браузера платформи), а як дані, над якими необхідно виконати дію, використано URI вебсторінки http://developer.android.com. Після цього платформа визначає активність, здатну опрацювати дію ACTION VIEW, і запускає її, передаючи зазначені дані.

Зазвичай описаним раніше способом здійснюють взаємодію зі стандартними компонентами платформи. За необхідності звернення до частин одного й того самого застосунку найчастіше використовують спрощений

підхід. Ідея його полягає в тому, що для ідентифікації необхідної активності використовують повну назву класу цієї активності.

Під час оголошення активності у файлі маніфесту елемент <intent-filter> не вказують:

```
<activity android:name="OtherActivity"
    android:label="@string/other activity name" />
```

Під завданням (task) в Android мають на увазі набір активностей, що викликають одна з одної та спрямованих на задоволення однієї потреби користувача. Список усіх завдань, що виконують на пристрої, відображають, коли користувач натискає й утримує кнопку Home.

Коли користувач запускає програму, створюють нове завдання й першу активність запущеної програми, що відкрилася, поміщають у стек активностей цього завдання. Щодо завдання цю активність називають *кореневою*. Завдання існує доти, доки коренева активність не завершиться.

Коренева активність може викликати другу активність, яку буде поміщено у стек поточного завдання поверх кореневої. Своєю чергою, друга активність може викликати третю тощо. Усі ці активності поміщають у стек. У разі закриття активності, що міститься на вершині стека (за натисканням кнопки Back на Android-пристрої або програмно за допомогою виклику відповідного методу), її видаляють зі стека, а управління передають тій активності, яка міститься у стеку безпосередньо під видаленою.

Під час натискання кнопки Home поточна активність переходить у фон, проте весь стек активностей відповідного завдання збережено. Якщо тепер натиснути й утримувати кнопку Home, то користувач побачить список активних завдань. У разі вибору будь-якої з них активність, що міститься на вершині стека, отримає фокус, а всі інші активності будуть зберігати у стеку, доки активи, що містяться вище, не будуть закрито.

Слід зазначити, що активності, що входять до стека певного завдання, можуть входити до складу різних застосунків. У цьому висловлено одну з важливих концепцій Android, яка декларує кооперацію кількох різних застосунків для задоволення потреб користувача та надає механізм інтентів для здійснення цієї кооперації.

Інтент, за допомогою якого запускають активність, є також посередником між активністю, що запускає й запускають. Його можна дістати, використовуючи виклик методу getIntent() у класі дочірньої активності. У визначеного об'єкта класу Intent можна дістати дію, категорії, URI та додаткові параметри, передані активністю, що викликає. Для цього використовують такі методи:

public String getAction();

public Set<String> getCategories();

public Uri getData();

public Bundle getExtras();

Описану можливість можна використовувати як для здобування даних, що потребують опрацювання цієї активності, так і для диспетчеризації дій у тому разі, коли одна активність здатна виконувати кілька різних дій.

Часто виникає необхідність передати інформацію не тільки від активності, що запускає, до тієї, що запускають, а й у зворотному напрямку. Типовий приклад – ситуація, коли дочірню активність використовують для введення даних, необхідних для головної активності. У цьому разі для запуску дочірньої активності необхідно використовувати не метод startActivity(), а метод startActivityForResult():

Intent intent = new Intent (this, EnterAddressActivity.class); startActivityForResult(intent, 1);

Тут наведено приклад запуску гіпотетичної активності під назвою EnterAddressActivity. Припустімо, що ця активність запитує в користувача адресу та під час натискання деякої кнопки завершує роботу й повертає введене значення головної активності. Для реалізації такого повернення значення в обробнику кнопки у класі EnterAddressActivity потрібен такий код (рис. 5.8).

```
String address = …; // retrieve address from the text field
Intent intent = new Intent(); intent.putExtra("address", address); setResult(RESULT
OK, intent); finish();
```

Рис. 5.8. **Код опрацювання кнопки у класі EnterAddressActivity**

Отже, створюють спеціальний інтент без указівки дії та класу, що містить здобуту від користувача адресу в додатковому параметрі (extra)

із ключем address. Далі за допомогою виклику setResult() указують, що виклик активності завершився успішно, після чого дочірню активність закривають за допомогою виклику методу finish().

Опрацювання повернення здійснюють у перевизначеному методі:

protected void onActivityResult (int requestCode, int resultCode, Intent data); головної активності. У цей метод як параметри resultCode і data передають значення, установлені дочірньою активністю, а як requestCode – ідентифікаційний параметр із виклику методу startActivityForResult() (у наведеному раніше прикладі – 1), який використовують для того, щоб розрізняти повернення з різних викликів – їхніх дочірніх активностей.

Практична складова до підрозділу 5
Робота з елементами та ресурсами Activity

Мета – набуття практичних навичок у програмуванні елементів та ресурсів Activity.

Програмна зміна властивостей

Створіть новий проєкт зі стандартними налаштуваннями з назвою **Lab2**.

Напишімо програму під умовною назвою "Світлофор". Інтерфейс програми буде мати такий вигляд. На червоному екрані розташовано три кнопки й один текстовий напис. Під час натискання кнопок фон програми буде змінюватися на відповідний колір, закріплений за певною кнопкою.

Створюймо новий проєкт на основі **Hello, World!**, видаляймо елемент TextView і перетягуймо з панелі інструментів дві кнопки Button.

У вікні **Component Tree** виділіть рядок **button**. У вас має з'явитися вікно властивостей **Properties**. Позбавімося від стандартних ідентифікаторів, а відразу будьмо привчати давати осмислені назви. Наприклад, для першої кнопки надаймо властивості id значення **buttonRed**, замість стандартного **@+id/button**. Для другої кнопки – значення **buttonYellow**.

У вікні властивостей Properties (рис. 5.9) змінімо **layout _ width** із **wrap _ content** (розмір за вмістом) на **match _ parent** (максимальний розмір, який допускає батьківський елемент activity).

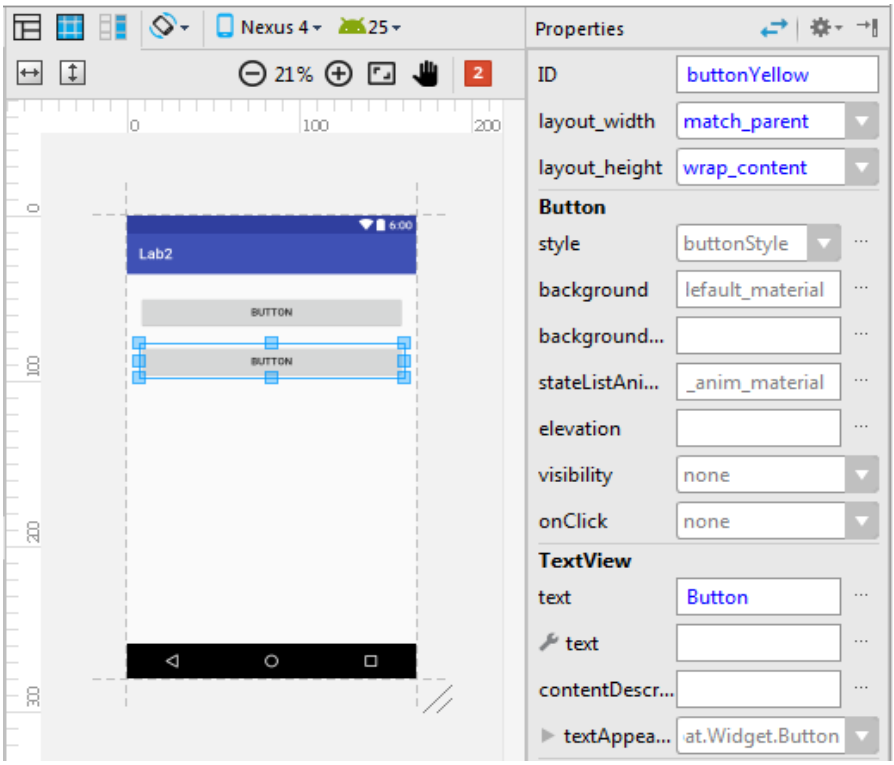


Рис. 5.9. **Зміни у вікні властивостей Properties**

Тепер створіть третю кнопку не через візуальне проєктування, а через код. Для цього в головному вікні слід перемкнутися із вкладки **Design** на вкладку **Text**. Тут побачите XML-розмітку програми, зокрема й код для двох кнопок.

Для створення третьої кнопки треба просто взяти за зразок код другої й додати його перед закривальним тегом **</RelativeLayout>**, як показано на рис. 5.10.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools" android:id="@+id/activity_main"
android:layout_width="match_parent" android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"

android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="ua.edu.uipa.ikpt.lab2.MainActivity">
```

Рис. 5.10. **Створення третьої кнопки**

```
<Button
    android:text="Button" android:layout_width="match_parent"
    android:layout_height="wrap_content" android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" android:layout_marginTop="10dp"
    android:id="@+id/buttonRed" />

<Button
    android:text="Button" android:layout_width="match_parent"
    android:layout_height="wrap_content" android:layout_below="@+id/buttonRed"
    android:layout_marginTop="10dp" android:id="@+id/buttonYellow" />

<Button
    android:text="Button" android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/buttonYellow" android:layout_marginTop="10dp"
    android:id="@+id/buttonGreen" />
    </RelativeLayout>
```

Закінчення рис. 5.10

Водночас слід не забути змінити ідентифікатор та атрибут **android:layout_below**.

Рядкові ресурси

Тепер треба замінити текст на кнопках на слова "червоний", "жовтий" і "зелений". На минулій лабораторній ми просто надали властивості Text потрібний рядок. Але, насправді, це неправильний підхід і середовище розроблення навіть виводить застережливі знаки. За правилами, рядки треба зберігати у рядкових ресурсах. Подібний підхід дає розробникові безліч переваг, зокрема, швидку локалізацію застосунку. Уважайте це стандартом, якого треба дотримуватися.

Процес створення рядкових ресурсів є дуже простим. Перейдіть назад у режим **Design** і виберіть кнопку **buttonRed**. У вікні властивостей знайдіть властивість **text**. Поруч міститься кнопка з трьома крапками. Клацніть на кнопці. У вас відкриється діалогове вікно **Resources** (рис. 5.11).

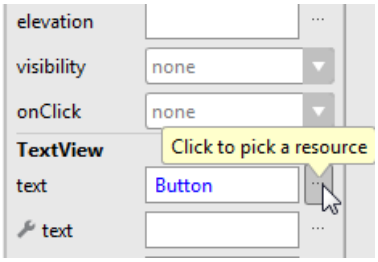


Рис. 5.11. **Діалогове вікно Resources**

Натисніть на випадний список **Add new resource** для створення нового рядкового ресурсу та виберіть **New String Value** (рис. 5.12).

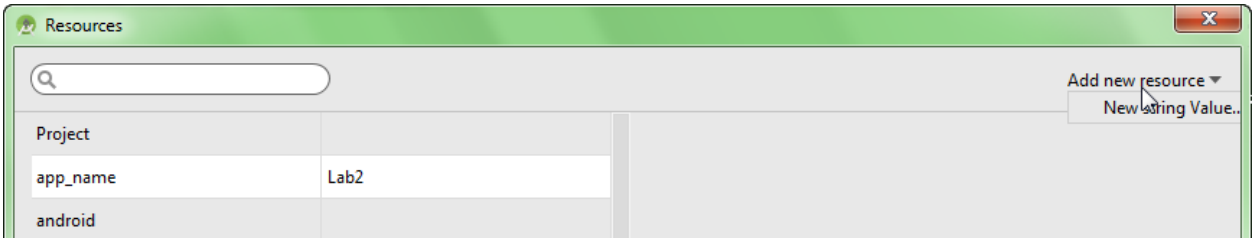


Рис. 5.12. **Налаштування Add new resource**

У новому вікні **New String Value Resource** (рис. 5.13) у першому полі **Resource Name** введіть назву ресурсу, наприклад, red, а у другому полі **Resource Value** введіть текст для кнопки (напр. червоний). Інші поля не чіпаймо. Аналогічним чином учиніть з іншими двома кнопками (жовтий і зелений).

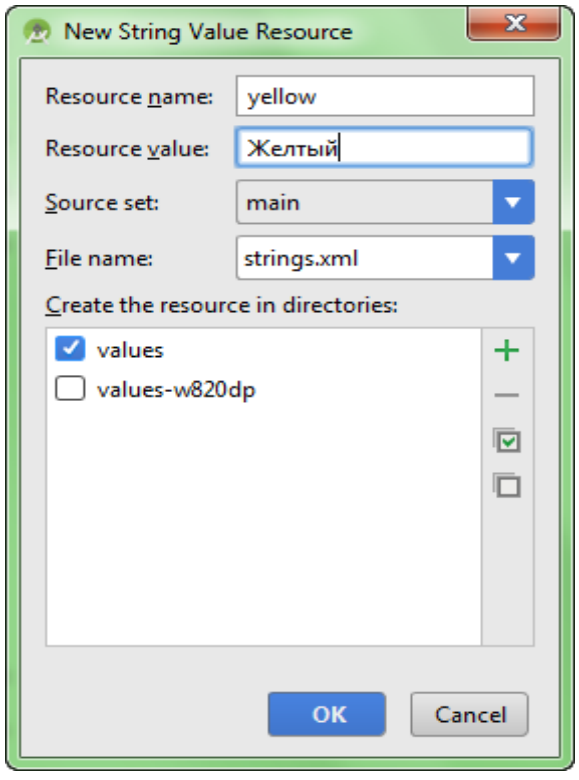


Рис. 5.13. **Налаштування New String Value Resource**

Програмно можна досягти такого самого результату, відредагувавши **файл strings.xml**, який міститься в теці **res/values** вашого проєкту. Зараз він має такий вигляд, як показано на рис. 5.14.

```
<resources>
    <string name="app_name">Lab2</string>
    <string name="red">Красный</string>
    <string name="yellow">Желтый</string>
```

Рис. 5.14. **Результат редагування файлу strings.xml**

Додаймо елемент **TextView** (а може, хто забув видалити створений за замовчуванням). Нехай на ньому виведено текст, що сповіщає про поточний колір фону застосунку. Оскільки в ресурсах у нас уже є слова "червоний", "жовтий" і "зелений" спочатку призначені для кнопок, то не будемо створювати нові рядкові ресурси, а скористаймося готовими напрацюваннями. За замовчуванням у нас використовують червоний колір. У вікні властивостей вибираймо властивість text для TextView і натискаймо кнопку із трьома крапками для виклику знайомого діалогового вікна. Цього разу не клацаймо на кнопці **New Resource**, а відразу виберімо рядок red, що, як пам'ятаємо, містить текст "червоний" і клацнімо кнопку OK (можна зробити відразу подвійне клацання на рядку).

Змініть **ID** для **TextView** на **textViewColor**. Перемкніться в текстовий режим і подивіться (рис. 5.15), який тепер буде мати вигляд опис для **TextView**.

```
<TextView
    android:text="@string/red" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textViewColor"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

Рис. 5.15. **Результат змінення TextView на textViewColor**

Текст на кнопках і тестовому полі дрібнуватий, змініть властивість **TextAppearance** на **AppCompat.Display1** через властивості або додайте код (рис. 5.16).

```
android:textAppearance="@style/TextAppearance.AppCompat.Display1" />
```

Рис. 5.16. **Результат змінення TextAppearance на AppCompat.Display1**

Рекомендовано постійно перемикатися в текстовий режим і дивитися, що відбувається в коді. Це дозволить упевненіше розбиратися в коді та читати чужий код. Зазвичай, новачки вважають за краще працювати через візуальні інструменти, а програмісти з досвідом самостійно пишуть практично весь код. Потрібно знайти розумний баланс між двома підходами. Усе прийде із часом.

Із ресурсами рядків ніби розібралися. Тепер у ресурсах задаймо колір для фону програми. Ресурси для кольорів прийнято зберігати в окремому файлі **colors.xml**, хоча технічно ніхто не забороняє зберігати їх у тому самому файлі **strings.xml**.

Відкриймо вказаний файл і додаймо ресурс червоного кольору між тегами resources (рис. 5.17).

```
<color name="colorRed">#FFFF0000</color>
```

Рис. 5.17. **Додавання ресурсу червоного кольору між тегами resources**

Зліва з'явиться червоний квадрат, на ньому легко бачити колір заданого ресурсу (рис. 5.18).

За таким самим принципом додайте жовтий і зелений колір.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="colorPrimary">#3F51B5</color>
<color name="colorPrimaryDark">#303F9F</color>
<color name="colorAccent">#FF4081</color>
<color name="colorRed">#FF0000</color>
<color name="colorYellow">#FFFF00</color>
<color name="colorGreen">#00FF00</color>
</resources>
```

Рис. 5.18. **Додавання кольору**

Визначивши в ресурсах всі необхідні кольори, можна відразу надати червоний колір для контейнера **RelativeLayout**. У вікні властивостей знаходьмо для цього елементу властивість **background** (щоб побачити всі

властивості компонента, натисніть посилання **View all properties**). Знову натискаймо кнопку із трьома крапками, щоб відкрити діалогове вікно. У вікні вибираймо розділ **Color**, шукаймо свій ресурс **color Red** (рис. 5.19).

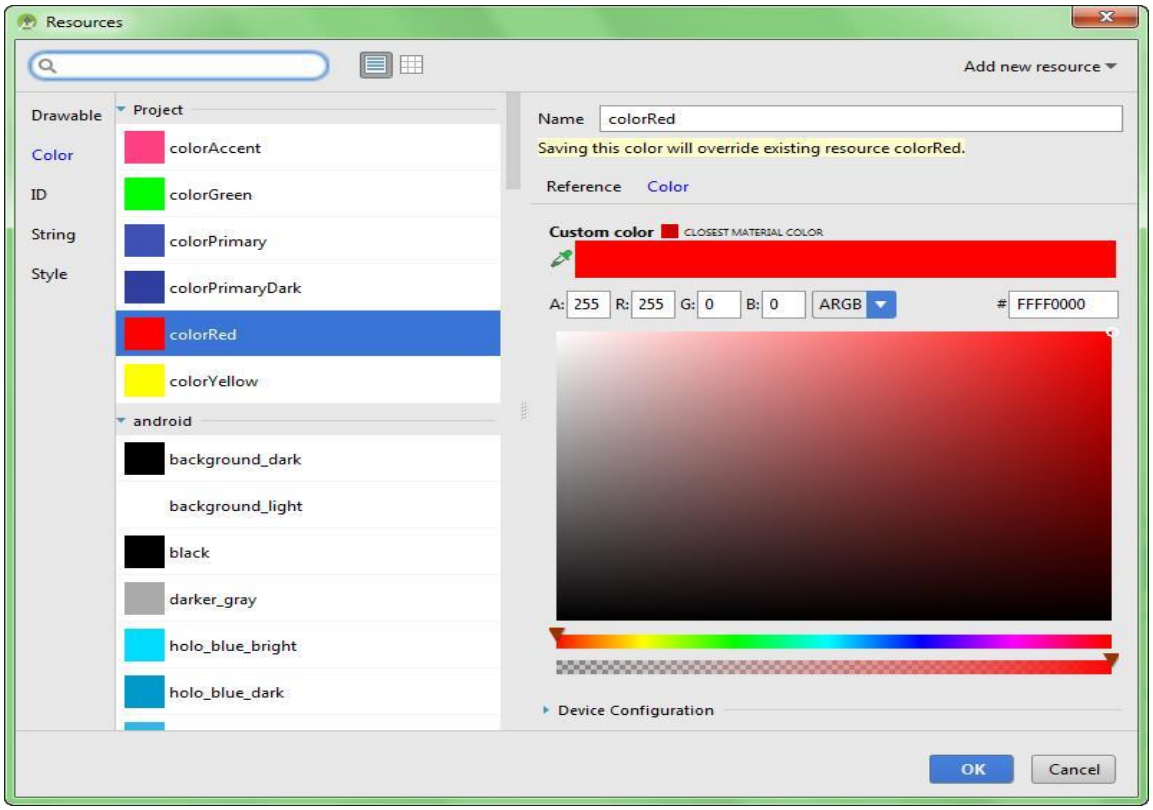


Рис. 5.19. **Вибір color Red**

Якщо подивитися в текстовому режимі, то побачите рядок android:background="@color/colorRed" для тега RelativeLayout.

Загальний каркас застосунку завершено. У нас є три кнопки з відповідними текстами, текстовий напис зі словом "червоний", і червоний фон, який використовують у контейнері RelativeLayout. Пора розпочинати програмну логіку програми. А поки можна запустити застосунок, щоб переконатися, що не зробили помилок у розмітці.

Опрацювання кнопок

Наше завдання – **опрацювати** клацання трьох кнопок і змінити колір фону застосунку, а також текст у **TextView**. На минулому занятті ми вже познайомилися зі зручним способом **опрацювання** події **onClick**.

Закріпімо пройдений матеріал і повторімо той самий код для червоної кнопки. Пропишімо вручну або додаймо через ресурси подію **onClick** в тезі **Button** (рис. 5.20).

```
android:onClick="onRedButtonClick"
```

Рис. 5.20. **Зміни подію onClick у тезі Button**

Далі в режимі **Text** поміщаймо курсор на назві методу та натискаймо комбінацію **Alt+ Ente**r, щоб створити заготовку клацання першої кнопки у класі **MainActivity**.

Оголосімо змінні для розмітки й текстового поля у класі та дістаньмо до них доступ у методі **onCreate ()**, як показано на рис. 5.21.

```
public class MainActivity extends AppCompatActivity {
    //до метода onCreate()
private RelativeLayout mRelativeLayout;private TextView mTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
    //в методе onCreate()

mRelativeLayout=(RelativeLayout) findViewById(R.id.activity_main);
mTextView=(TextView)findViewById(R.id.textViewColor);
}
```

Рис. 5.21. **Дістання доступу в методі onCreate ()**

Якщо копіюєте цей код із методички та вставляєте в себе, то назви класів буде виділено червоним кольором. Натискаймо комбінацію **Alt + Enter** та імпортуймо необхідні класи.

Також зверніть увагу, що звертаємося до компонентів RelativeLayout TextView за ідентифікаторами **activity_main** і **textViewColor**, які потрібно було прописати раніше в **activity_main.xml**. Надалі намагайтеся відразу надавати ідентифікатори елементів, із якими доведеться працювати в коді.

Пишімо код для клацання кнопки з написом "червоний" (рис. 5.22).

```
public void onRedButtonClick(View view) {
        mTextView.setText(R.string.red);

mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.colorRed));
}
```

Рис. 5.22. **Код для клацання кнопки з написом "червоний"**

AndroidStudio посилає на метод getColor, але працює. Справа в тому, що в Android 6 (API 23) метод **getColor** (int id) оголосили застарілим і студія тепер підкреслює цей метод. В API 23 і пізніх версіях можна замінити на варіант, показано на рис. 5.23.

```
        // null mRelativeLayout.setBackgroundColor(getResources().getColor(R.colo
r.yellowColor, null));
```

Рис. 5.23. **Код зміни методу getColor (int id)**

Звертаймося до створених ресурсів через спеціальний клас Ri, через точку вказуймо тип ресурсів, а потім назву ресурсу.

Для кнопок "зелений" і "жовтий" напишіть код самостійно, додавши метод onGreenButtonClick () та onYellowButtonClick () в activity_main.xml.

Запускаймо програму та клацаймо по кнопках – текст у написах і фон у застосунку має змінюватися, відповідно до натиснутої кнопки (рис. 5.24).

Рис. 5.24. **Результат змінення тексту в написах та фону в застосунку**

Повний текст коду буде мати такий вигляд, як показано на рис. 5.25.

```
package ua.edu.uipa.ikpt.lab2;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    //до метода onCreate()
    private RelativeLayout mRelativeLayout;private TextView mTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //у методі onCreate()
mRelativeLayout=(RelativeLayout)findViewById(R.id.activity_main);
        mTextView=(TextView)findViewById(R.id.textViewColor);
    }
    public void onRedButtonClick(View view) {
        mTextView.setText(R.string.red);
mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.colorRed));
    }

    public void onYellowButtonClick(View view) {
            mTextView.setText(R.string.yellow);
        public void onGreenButtonClick(View view) {
        mTextView.setText(R.string.green);

mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.colorGreen));
    }

mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.colorYellow));
    }
}
```

Рис. 5.25. **Програмний код змінення тексту в написах та фону в застосунку**

Застосунок не завжди складається з одного екрана. Наприклад, створено дуже корисну програму й користувачеві хочеться дізнатися, хто ж її автор. Він натискає на кнопку "Про програму" й потрапляє на новий екран, де міститься корисна інформація про версії програми, автора,

адресу сайта тощо. Сприймайте екран активності як вебсторінку з посиланням на іншу сторінку. Якщо подивитися на код у файлі **MainActivity.java**, то побачите, що клас **MainActivity** також належить до Activity або, якщо говорити точніше, його успадковано від нього (рис. 5.26).

```
public class MainActivity extends AppCompatActivity
```

Рис. 5.26. **Фрагмент коду у файлі MainActivity.java**

Слід створити новий клас, який може бути схожим на **MainActivity** і потім якось перемкнутися на нього в разі натискання кнопки.

Виберіть у меню **File | New | Activity | Empty Activity**. Далі з'явиться знайоме вам вікно створення нової активності. Заповнюймо необхідні поля (рис. 5.27).

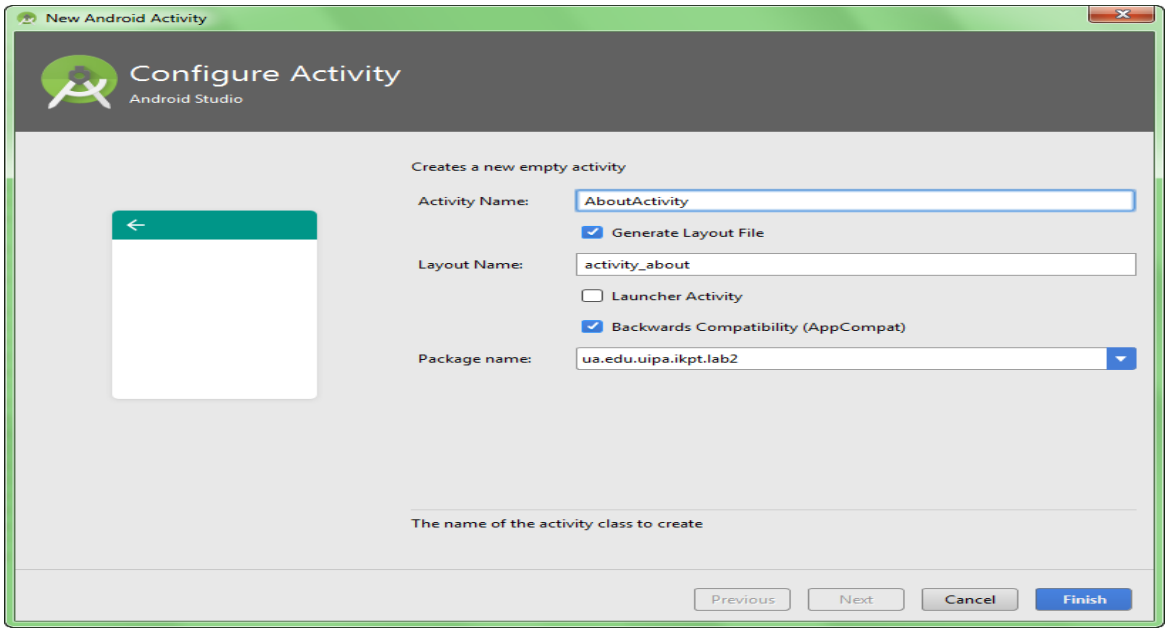


Рис. 5.27. **Вікно створення нової активності**

Натискаймо на кнопку **Finish** і активність буде готовою. Щоб переконатися в цьому, відкрийте файл маніфесту й перевірте наявність нового запису. Файли класу **AboutActivity** та розмітки **activity_about** мають відкриватися.

Вийде відповідна заготовка, у яку вставмо елемент **TextView**, призначімо для нього ID – **textViewAbout**. У тексті або через ресурси змініть

layout_width і **layout_height** на **match_parent**. Відцентруйте елемент TextView (рис. 5.28).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_about" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="ua.edu.uipa.ikpt.lab2.AboutActivity">
    <TextView
        android:text="TextView" android:layout_width="match_parent"
        android:layout_height="match_parent"android:id="@+id/textViewAbout"
```

Рис. 5.28. **Код опрацювання елементу TextView**

Тепер оформімо вміст вікна About через ресурси. Відкриймо файл **res /values/strings.xml** і вводьмо такий текст вручну (рис. 5.29).

```
<string name="about_text">
<b>Світлофор</b>\версія
0.1\n\n Розробник:\n
<b>Левченко М.О.</b>\n
</string>
```

Рис. 5.29. **Код файлу res/values/strings.xml**

Можна використовувати найпростіші HTML-теги форматування тексту типу <b>, <i>, <u>. Для перекладу тексту на новий рядок використовуйте символи **\n**. Додаймо ще один рядковий ресурс для заголовка нового екрана (рис. 5.30).

```
<string name="about_title">О программе</string>
```

Рис. 5.30. **Додавання рядкового ресурсу для заголовка нового екрана**

Повернімося в **activity_about.xml** і замінімо посилання на ресурс, також збільшімо розмір тексту через **textAppearance** (рис. 5.31).

```
android:text="@string/about_text"
android:textAppearance="@style/TextAppearance.AppCompat.Display1 "
```

Рис. 5.31. **Код збільшення розміру тексту через textAppearance**

Тепер наше завдання – перейти на новий екран під час натискання кнопки мишки на першому екрані. Створюймо кнопку **buttonAbout** і через ресурси розміщуймо на ній текст "Про програму". Створюймо обробник клацання на цю кнопку **onClickAbout**.

Переходьмо назад до класу **MainActivity**. Для запуску нового екрана необхідно створити екземпляр клас **aIntenti** вказати в першому параметрі поточний клас, а у другому – клас для переходу, у нас це **AboutActivity**. Після цього викликають метод **startActivity ()**, який і запускає новий екран.

Пишімо обробник клацання кнопки (рис. 5.32).

```
public void onClickAbout(View view) {

Intent intent=new Intent(MainActivity.this, AboutActivity.class);
startActivity(intent);
}
```

Рис. 5.32. **Код обробника клацання кнопки**

Новий **Activity** вже зареєстровано в маніфесті **AndroidManifest.xml**. Знайдіть цей файл у своєму проєкті та двічі клацніть на ньому. Відкриється вікно редагування файлу. Додайте посилання з label на ресурс **about_title**. Друкуйте самостійно й активно використовуйте підказки. Вийде таке (рис. 5.33).

```
<activity android:name=".AboutActivity"
android:label="@string/about_title">

</activity>
```

Рис. 5.33. **Новий ресурс Activity**

Ось і знадобився рядковий ресурс **about_title**. Запускаймо програму, клацаймо на кнопці й маємо вікно програми (рис. 5.34). Отже, ми навчилися створювати нове вікно та викликати його одним натисканням кнопки.

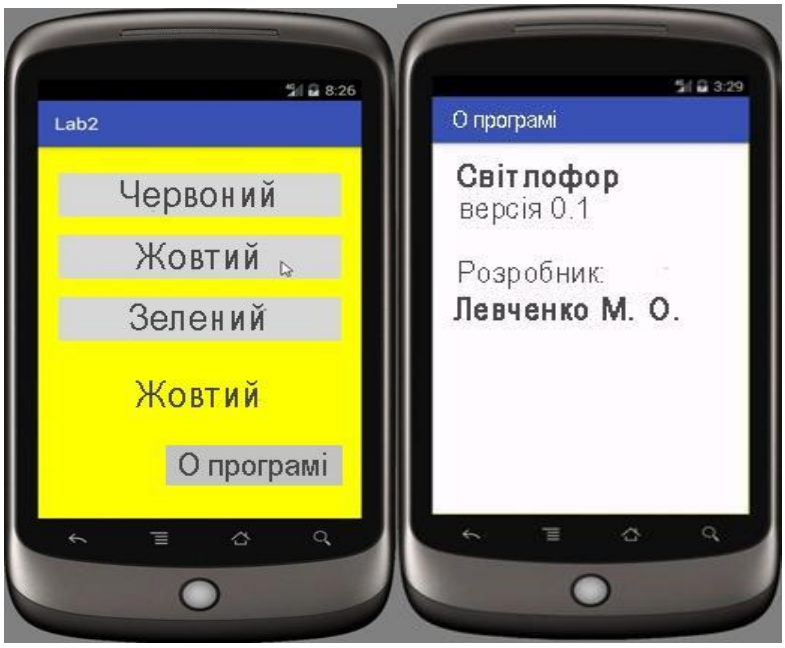


Рис. 5.34. **Результат запуску програми**

Звертаймо увагу, що другий створюваний клас активності містить успадковувати від класу **Activity** йому подібних (ListActivityi ін.), мати XML-файл розмітки (якщо потрібно) і бути прописаним у маніфесті.

Після виклику методу **startActivity ()** запуститься нова активність (у цьому разі **AboutActivity**), вона стає видимою та переміститься на вершину стека, що містить компоненти, що працюють. Під час виклику методу **finish ()** із нової активності (або натискання клавіші апаратного повернення) її буде закрито та видалено зі стека. Розробник також може переміщатися до попередньої (або до будь-якої іншої) активності, використовуючи той самий метод **startActivity ()**.

Виправлення помилок

Спершу всі роблять помилки – помилки копіювання фрагмента коду без його розуміння, пишуть не в тому місці тощо. У редакторі коду у верхньому правому куті є прямокутник. Він може бути зеленим (ідеальний код), жовтим (не смертельно, але краще виправити) і червоним (помилка в коді, програма не запуститься).

Ваше завдання – прагнути до зеленого кольору. Жовтий колір бажано переглядати ті вирішувати самостійно, чи потрібно виправляти код. Якщо розумієте, у чому проблема, то виправте. Якщо не зрозуміло, то залиште. Попередження не завжди бувають доречними, іноді їх можна ігнорувати. Розуміння прийде з досвідом і практикою.

Далі на рис. 5.35 показаний приклад, коли розробник забув поставити крапку з комою в кінці рядка.

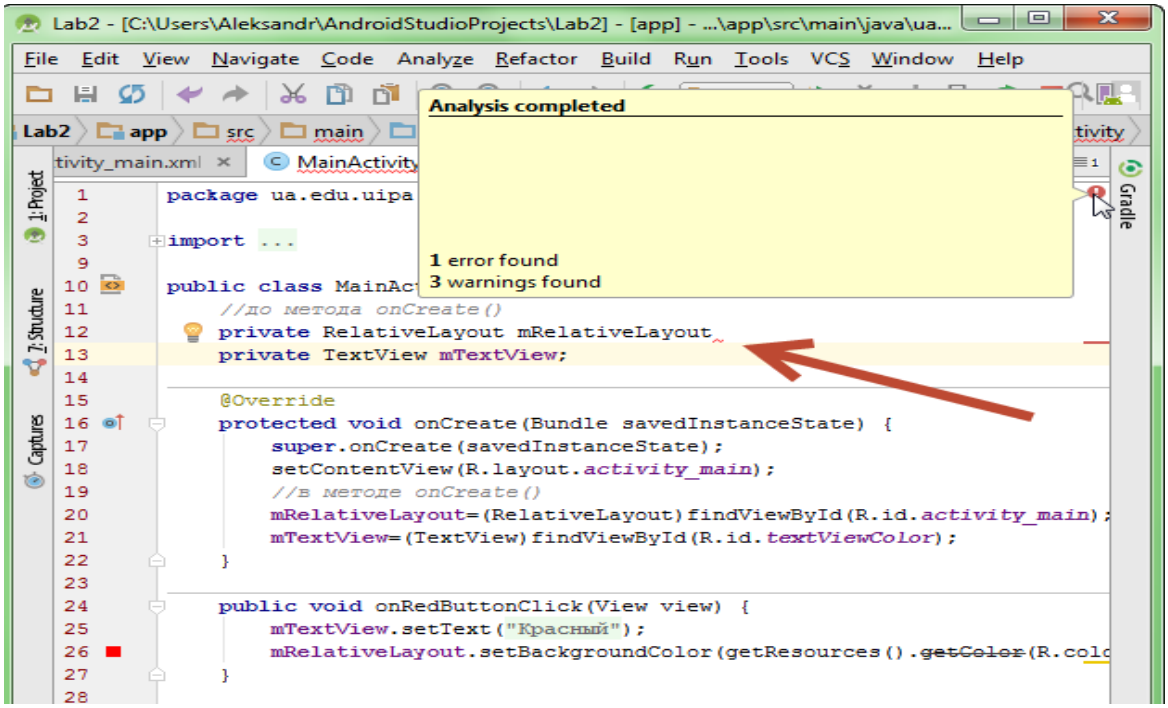


Рис. 5.35. **Приклад, коли розробник забув поставити крапку з комою в кінці рядка**

Крім прямокутника, там далі є зарубки з такими саме кольорами. Підведіть курсор мишки до будь-якої із зарубок і побачите підказку про характер помилки або попередження. Клацання на зарубку перенесе у потрібне місце в документі.

Поговорімо про значки. За замовчуванням студія використовує зображення зеленого робота як знак для вашої програми. Відкрийте у студії папку **res/mipmap**. Ця папка є віртуальною та, насправді, є папки з назвами **res/drawable-hdpi, res/drawable-mdpi, res/drawable-xhdpi, res/drawable-xxhdpi**. У кожній із цих папок є файл з однаковою назвою **ic_launcher.png**. Уся різниця між цими файлами полягає в розмірах. Залежно від роздільної здатності екрана, на пристрої система вибирає найбільш відповідне за розміром зображення та виводить його як значок у заголовку програми та на домашньому екрані. Найпростіший варіант замінити стандартне зображення на своє – створити своє зображення й замінити його наявний значок. Рекомендовано створювати під кожну роздільну здатність свій значок. Причому тут указано не всі варіанти. У такому разі потрібно створити самостійно папку, наприклад, **drawable-xxxhdpii**, розмістити там картинку необхідного розміру. Якщо пропустите

якийсь розмір, то система спробує взяти який-небудь значок із цією назвою з іншої папки та масштабувати його. Але краще так не робити.

Якщо не хочете змінювати наявні стандартні значки, а використовувати значки під іншою назвою, то в цьому разі підготуйте всі необхідні розміри, розмістіть їх у всіх папка **xdrawable** під своєю назвою, а потім у маніфесті (**manifests -> AndroidManifest.xml**) замініть рядок в атрибута **android**: icon.

До складу студії входить набір зумовлених значків і генератор власних значків. Щоб його побачити, клацніть правою кнопкою на папці mipmap і виберіть у меню **New | Image Asset**.

Відкриється діалогове вікно, де можете вказати як джерело файл на комп'ютері, варіант із кліпарта або набір символів. Також можете задати форму значка, колір фону та інші параметри. Самостійно замініть іконку на одну із **ClipArt** (рис. 5.36).

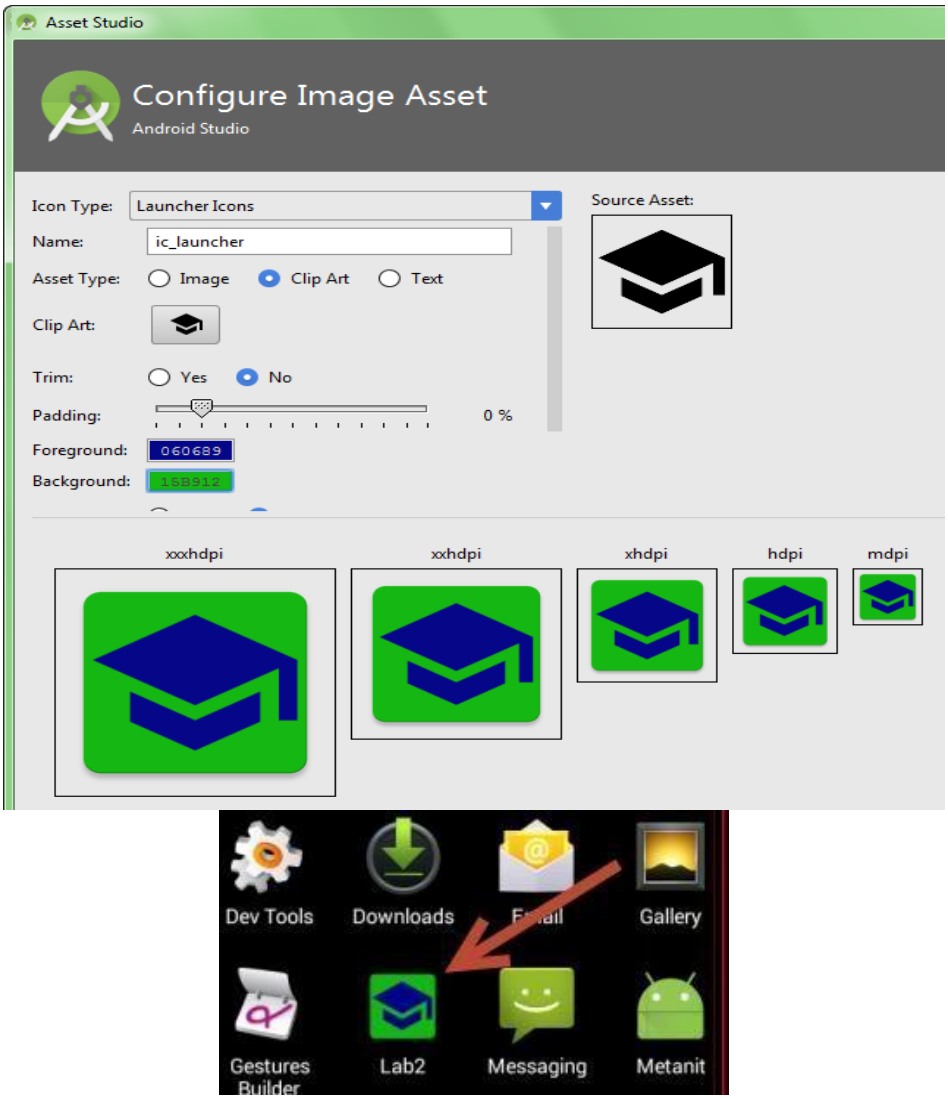


Рис. 5.36. **Заміна іконки на одну з іконок ClipArt**

Крім значків для різних роздільних здатностей, генератор створить додатковий файл із суфіксом **web**, який буде скопійованим у папку **main**. Цей файл використовуйте для Google Play, коли будете розміщувати застосунок у магазині застосунків і давати опис до нього.

Контрольні запитання для самоперевірки

1. Що таке "ресурси"? Для вирішення яких завдань розроблено цей механізм Android? Які переваги дає розробнику використання механізму ресурсів?

2. Які є типи ресурсів? Як розміщено у проєкті файли ресурсів?

3. Як можна використовувати ресурси в застосунку безпосередньо із програмного коду, а також з інших ресурсів?

4. Що таке ресурси, які види конфігурації? Для чого призначено цей механізм і як його можна використати?

5. Як, використовуючи механізм ресурсів, створити головне меню або панель дій застосунків Android?

6. У чому полягає різниця в реалізації меню для ранніх та пізніх версій платформи Android?

7. Як опрацювати вибір дій із головного меню чи панелі дій?

Рекомендована література: [1; 4; 5; 6; 10 – 13; 16].

6. Зберігання даних на платформі Android

Мета – дослідження специфіки зберігання даних на платформі Android.

Професійні компетентності: здатність застосовувати сучасні методи й інструменти для досліджень у сфері видавництва та поліграфії, а також забезпечення якості продукції; здатність аналізувати структуру та контент проєктів інтерактивних медіа.

6.1. Методи зберігання даних

Багатьом програмам потрібно зберігати дані в постійній пам'яті та відновлювати їх під час наступних запусків. Платформа Android надає

три можливості вирішення цього завдання: налаштування (prefereneces), файлову систему та базу даних. Розгляньмо ці можливості докладніше.

Механізм налаштувань. Як випливає з назви, основне призначення цього механізму полягає у зберіганні налаштувань застосунку. Android API дозволяє зберігати налаштування у вигляді пар "ключ – значення" й автоматично вирішує всі завдання створення та управління файлами, у яких ці налаштування зберігають.

Для того щоб почати працювати з налаштуваннями, необхідно визначити об'єкт класу SharedPreferences за допомогою виклику методу public SharedPreferences getSharedPreferences(String name, int mode); класу Context. Як перший аргумент передають ідентифікатор файлу налаштувань. Другий аргумент визначає режим доступу. Здебільшого достатньо використовувати режим за замовчуванням (Context.MODE PRIVATE). Інші значення цього параметра дозволяють створювати налаштування, що поділяють кількома застосунками. Про них можна докладніше дізнатися з документації.

Після того як об'єкт класу SharedPreferences отримано, можна вилучати записані раніше налаштування за допомогою методів:

public String getString(String key, String defaultValue);

public int getInt(String key, int defaultValue);

Кожен із параметрів ідентифікують за ключом (параметр **key**). Якщо запитаного значення у файлі налаштувань не надано, то повертається значення за замовчуванням – його передають у виклик методу get.() як другий аргумент.

Ідентифікатор може бути будь-яким рядком, проте необхідно вживати заходів щодо забезпечення унікальності цього рядка. Наприклад, для налаштувань, що стосуються однієї конкретної активності, розумно використовувати як ідентифікатор повну назву класу цієї активності.

У класі визначено методи для кожного із примітивних типів та типу String.

Збереження налаштувань здійснювати є трохи складнішим. Спочатку необхідно визначити об'єкт класу, що реалізує інтерфейс Shared Preferences. Editor за допомогою виклику методу, що дозволяють зберігати налаштування відповідних типів.

public SharedPreferences.Editor edit();

класу SharedPreferences. Визначений об'єкт має такі методи:

public SharedPreferences.Editor putString(String key, String value);

public SharedPreferences.Editor putInt (String key, int value);

Якщо значення параметра, що відповідає ключу, за яким здійснено запис, уже було у файлі налаштувань, це значення перезаписують. Негайно після закінчення запису даних необхідно викликати метод public boolean commit(), який атомарно збереже зміни у файлі налаштувань.

Проілюструймо застосування механізму налаштувань на наступному прикладі. Розгляньмо застосунок "Лічильник" і додаймо в нього можливість зберігати стан лічильника між запусками застосунку. Незважаючи на те що стан не є, узагалі кажучи, налаштуванням застосування, використання такого механізму в цьому разі є найпростішим вирішенням завдання і тому є доречним.

Будемо здійснювати збереження стану в методі onPause(), а відновлення – у методі onResume(). Для того щоб забезпечити відновлення стану, визначмо метод установлення значення у класі Counter (рис. 6.1):

```
public void setValue(int value)
    this.value = value;
if (listener != null) { listener.onModification(this); }
}
```

Рис. 6.1. **Метод установлення значення у класі Connter**

Із класу MainActivity необхідно видалити код збереження та відновлення стану лічильника в разі повороту, оскільки вирішення цього завдання в новій версії програми буде досягнуто автоматично. Повний код модифікованого класу показано на рис. 6.2.

```
public class MainActivity extends Activity
  private TextView counterText;
  private Counter counter = new Counter();
  @Override
  public void onCreate(Bundle savedInstanceState)
    super.onCreate(savedInstanceState); setContentView(R.layout.main);
    counterText = (TextView) findViewById(R.id.counterText);
    counter.setOnModificationListener(
        new Counter.OnModificationListener() @Override
      public void onModification(Counter sender) { updateCounterView(); }
    });
  }
```

Рис. 6.2. **Повний код модифікованого класу**

```
public void updateCounterView ()
counterText.setText(String.valueOf(counter.getValue()));
}
public void onIncreaseButtonClick(View v) counter.increase();
}
public void onResetButtonClick(View v) counter.reset();
}
@Override
protected void onPause()
  super.onPause();
  SharedPreferences prefs = getSharedPreferences(getLocalClassName(),
      Context.MODE PRIVATE);
  SharedPreferences.Editor editor = prefs.edit();
  editor.putInt("counterValue", counter.getValue());
  editor.commit();
}
```

```
                                    @Override
                          {         protected void onResume()
                                        super.onResume();
    SharedPreferences prefs =getSharedPreferences(getLocalClassName(),
                      `                 Context.MODE PRIVATE);
                        counter.setValue(prefs.getInt("counterValue", 0));
                                                    }}
```

Закінчення рис. 6.2

Основні класи до роботи системи управління базою даних (СУБД) SQLite. SQLite є вбудовуваною СУБД, що за замовчуванням підтримано в Android. Її використання у програмах ґрунтується на застосуванні двох класів Android API: SQLiteDatabase та SQLiteOpenHelper. Перший інкапсулює операції доступу до бази даних (БД), включно з додаванням, зміною, видаленням даних із таблиць, запитами на вибірку даних, а також управлінням структурою БД. Другий клас є допоміжним і призначеним для управління життєвим циклом БД, включно з початковим створенням схеми даних та оновленням цієї схеми під час оновлення програми.

Розробник Android-застосунку сам відповідальний за те, щоб необхідну програму базу даних було створено перед початком використання і вона мала актуальну версію. Щоб гарантувати це, слід створити клас, успадкований від абстрактного класу SQLiteOpenHelper, визначити конструктор і такі методи:

public void onCreate(SQLiteDatabase db);

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion);

Конструктор успадкованого класу, зазвичай, просто викликає конструктор суперкласу:

```
SQLiteOpenHelper(Context context, String name,
    SQLiteDatabase.CursorFactory factory, int version);
```

Із відповідними значеннями параметрів, як-от **context** передають поточний контекст (можна просто передати посилання на об'єкт класу активності), **name** містить назву бази даних, **version** – номер версії БД, а **factory** зазвичай установлюють null.

Метод **onCreate()** викликають у тому разі, коли відбувається перше з моменту встановлення програми звернення до БД. Типова реалізація цього методу полягає у виконанні SQL-команди create, що створює схему даних, за допомогою виклику методу public void execSQL(String sql); на об'єкті класу SQLiteDatabase.

Метод **onUpdate()** призначено для внесення змін до БД під час оновлення програми. Його викликають у тому разі, якщо номер поточної версії бази даних у системі не збігається із числом, переданим як параметр version у конструктор класу SQLiteOpenHelper. Номери старої та нової версій передають методом onUpdate() як аргументи. На підставі їхнього порівняння програміст може виконати команди зміни схеми даних, що перетворюють схему даних від старої версії до нової.

Зазвичай, об'єкт класу, успадкований від SQLiteOpenHelper, створюють у методі onCreate() і розміщують у полі класу активності.

Надалі на цьому об'єкті викликають метод public SQLiteDatabase getWritableDatabase(); що повертає об'єкт класу SQLiteDatabase, через який здобувають доступ до даних. Після закінчення використання необхідно закрити БД за допомогою виклику методу public void close(); на об'єкті класу SQLiteOpenHelper.

6.2. Доступ до даних

Клас **SQLiteDatabase** надає безліч методів доступу до даних. Розгляньмо основні з них.

Метод public long insert(String table, String nullColumnHack, ContentValues values); призначено для вставлення рядка до таблиці БД. Назву таблиці передають як параметр **table**, а значення, що вставляють, як параметр **values**. У результаті виклику методу повертається кількість доданих рядків або -1, якщо додавання не вдалося здійснити.

Клас **ContentValues**, що використовують для зберігання значень параметрів, є асоціативним масивом, у якому ключами є назви стовпців

таблиці, а значеннями дані відповідних клітинок. Для запису пари "ключ – значення" в асоціативний масив використовують метод put() аналогічно стандартним асоціативним масивам Java.

Метод public int update(String table, ContentValues values, String whereClause, String[] whereArgs); призначено для зміни запису у БД. Параметри table і values мають такий самий зміст, як і в методі insert(). Параметр whereClause містить вираз на мові SQL, що здійснює вибірку записів для оновлення. Цей вираз може містити параметри підстановки, що позначено знаками запитання. Значення цих параметрів передають у вигляді масиву, де Args у тому порядку, у якому вони зустрічалися в SQL-виразі.

Метод public int delete (String table, String whereClause, String[] whereArgs); призначено для видалення записів із бази даних. Усі параметри є аналогічними вже розглянутим.

Для здобуття даних із однієї таблиці бази даних використовують такий метод:

Public Cursor query(String table, String[] columns, String selection,
String[] selectionArgs, String groupBy, String having,
String orderBy, String limit);

Параметри методу означають таке:

• table – назва таблиці, із якої здійснюють вибірку;

• columns – список стовпців, які слід повернути в результаті запиту (значення null означає всі стовпці);

• selection – SQL-вираз для конструкції where SQL-запиту на вибірку (може містити параметри підстановки);

• selectionArgs – значення параметрів підстановки;

• groupBy, having, orderBy, limit – SQL-вирази для конструкцій group by, having, order by та limit запиту на вибірку даних. Більшість параметрів є необов'язковими й можуть містити значення null.

У разі, коли необхідно здобути дані з кількох таблиць, використовують більш загальний метод public Cursor rawery(String sql, String[] selectionArgs).

Цей метод приймає SQL-запит, який безпосередньо передає СУБД на виконання.

У результаті виконання методів query() та rawery() повертається об'єкт класу, що реалізує інтерфейс Cursor, який призначено для навігації за результівним набором даних.

Спочатку курсор міститься в позиції, що передує першому рядку набору даних. Для переходу до наступної позиції використовують метод

public boolean moveToNext(); який повертає true, якщо перехід був успішним і false, якщо набір даних закінчився.

Для визначення полів рядка, через який "переступив" курсор, використовують один із таких методів:

public int getInt(int columnIndex);

public long getLong(int columnIndex);

public String getString(int columnIndex);

залежно від типу визначених даних. Як параметр columnIndex у наведені раніше методи передають порядковий номер стовпця, значення якого потрібно визначити. За необхідності номер стовпця результівного набору даних, а також його тип можна отримати на назву, використовуючи такі методи:

public int getColumnIndex(String columnName);

public int getType(int columnIndex);

Допустимі типи даних визначено як статичні константи інтерфейсу Cursor.

Розгляньмо приклад. Показана на рис. 6.3 далі функція виводить уміст довільної таблиці у файл журналу (це може бути корисним для налаштування програми).

```
private void printTable(SQLiteDatabase database, String tableName) Cursor
  cursor = database.query(tableName, null, null, null, null,
     null, null, null);
  while                  {                  (cursor.moveToNext())
    Log.d(getLocalClassName(), "Record:");
    for (int i = 0; i < cursor.getColumnCount(); i++) {

                String columnName = cursor.getColumnName(i) + ": ";
                          swit (cursor.getType(i))
                  case Cursor.FIELD TYPE INTEGER:
              Log.d(getLocalClassName(), columnName + cursor.getInt(i));
                          break;
                  case Cursor.FIELD TYPE STRING:
            Log.d(getLocalClassName(), columnName + cursor.getString(i));
                          break;
                     }
                     }
                     }
                     }
```

Рис. 6.3. **Функція виведення вмісту довільної таблиці у файл журналу**

Для простоти передбачають, що стовпці таблиці можуть мати цілий чи рядковий тип.

6.3. Асинхронне виконання

Асинхронне виконання передбачає використання окремих потоків для виконання деяких дій. Необхідність в асинхронному виконанні найчастіше викликано наявністю в застосунку деяких процесів (обчислення, звернення до мережевих ресурсів, читання з бази даних), що потребують досить великих витрат часу. Якщо не переносити виконання цих процесів на окремі потоки, це неминуче позначиться на чуйності інтерфейсу користувача, знижуючи зручність програми.

Організація асинхронного виконання може спиратися на стандартний механізм потоків Java (клас read та інтерфейс Runnable) або використовувати Android-специфічні API, які є високорівневими обгортками стандартних потоків.

Важливою проблемою асинхронного виконання є синхронізація потоків. Необхідність її пов'язано з тим фактом, що засоби Android API, як і будь-якої бібліотеки графічного інтерфейсу, є небезпечними. Останнє означає, що звернення до віджетів із потоків, відмінних від головного потоку виконання програми, може призводити до непередбачуваних наслідків.

Для розв'язання проблеми синхронізації Android API надає різні засоби, що містять черги повідомлень, клас для організації виконання асинхронних завдань AsyncTask, а також спеціалізовані механізми, орієнтовані на вирішення окремих завдань, як-от асинхронне завантаження даних із БД.

Клас Handler призначено для управління чергами повідомлень, пов'язаних із потоками виконання. Повідомлення можна надсилати в чергу з будь-якого потоку виконання, але опрацьовують їх завжди на головному (пов'язаному з інтерфейсом користувача) потоці. Отже, клас Handler забезпечує синхронізацію потоків.

Для надсилання повідомлення в чергу використовують такі методи:

```
public boolean sendMessage(Message msg);
 public boolean sendMessageDelayed(Message msg, long delayMillis);
 public boolean sendMessageAtTime(Message msg, long uptimeMillis);
```

Повідомлення є об'єктом класу Message. Для зберігання деталей повідомлення, що передають, можуть використовувати цілочислові властивості з назвами what, arg1 і arg2, а також властивість obj типу Object. Android API не регламентує спосіб застосування цих властивостей, тому розробник може використовувати їх на власний розсуд.

Для оптимізації використання оперативної пам'яті рекомендовано не створювати об'єкти класу Message за допомогою операції new, а викликати один зі статичних методів Handler:

public Message obtainMessage(int what, int arg1, int arg2, Object obj);

public Message obtainMessage(int what)

public Message obtainMessage(int what, Object obj);

Ці методи забезпечують повторне використання об'єктів повідомлень, організовуючи пул. У разі запиту об'єкта із значеннями параметрів, що повторюють, ці методи просто повертають його з пулу, а не створюють новий об'єкт.

Для опрацювання повідомлень необхідно успадкувати власний клас від класу Handler, перевизначити метод public void handleMessage(Message msg); і в ньому розмістити код опрацювання повідомлення, що передають у метод як аргумент. Опрацювання повідомлення здійснюють на головному потоці виконання програми, тому із цього обробника можна звертатися до елементів інтерфейсу користувача.

Крім надсилання повідомлень у чергу, можна додавати об'єкти класів, що реалізують інтерфейс Runnable. Це здійснюють за допомогою таких методів:

public boolean post (Runnable r);

public boolean postDelayed(Runnable r, long delayMillis);

public boolean postAtTime (Runnable r, long uptimeMillis);

Під час використання цього підходу оброблювачем є метод run() об'єкта, що передають.

Як приклад використання класу Handler розгляньмо програму, яка визначає зовнішню IP-адресу пристрою за допомогою онлайн-сервісу Google. Оскільки виконання мережевого запиту потребує часу, украй небажано виконувати подібні дії на основному потоці виконання. Для розв'язання цієї проблеми створімо окремий потік, який буде виконувати запит, а потім передавати визначену в результаті запиту до сервісу

IP-адресу на головний потік програми. Інтерфейс програми містить кнопку для ініціювання запиту, індикатор прогресу та поле для виведення результату. Файл res/layout/main.xml, що описує інтерфейс головної активності, має такий вигляд (рис. 6.4).

```
<?xml version="1.0" encoding="utf 8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/
android" android:orientation="vertical"
android:layout width="fill parent" android:layout
height="fill parent">
<LinearLayout android:orientation="horizontal"
android:layout width="wrap content"
android:layout height="wrap content"
android:layout gravity="left center vertical">
<Button android:layout width="wrap content"
android:layout height="wrap content"
android:text="Determine IP address"
android:onClick="onDetermineIPAddressClick"/>
ProgressBar android:id="@+id/progressBar"
android:layout width="wrap content"
android:layout height="wrap content"
android:visibility="invisible"/>
</LinearLayout>
TextView android:id="@+id/ipTextView"
android:layout width="wrap content"
android:layout height="wrap content"
android:layout gravity="left center vertical"/>
</LinearLayout>
```

Рис. 6.4. **Файл res/layout/main.xml**

Оскільки для правильного функціонування застосунку потрібно мати доступ до інтернету, необхідно додати відповідний дозвіл у файл маніфесту:

```
<uses−permission android:name="android.permission.INTERNET" />
```

Ініціалізація класу головної активності є стандартною, у методі onCreate() відбувається заповнення полів класу активності, що зберігають посилання на віджети інтерфейсу користувача (рис. 6.5).

```
public class MainActivity extends Activity
private Handler handler;
private TextView ipTextView;
private ProgressBar progressBar;
@Override
public void onCreate(Bundle savedInstanceState)
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
ipTextView = (TextView) findViewById(R.id.ipTextView);
progressBar = (ProgressBar) findViewById(R.id.progressBar);
handler = new Handler()
@Override
public void handleMessage(Message msg)
handleIPDeterminationMessage(msg);
}
};
// …}
```

Рис. 6.5. **Ініціалізація класу головної активності**

У полі handler записують об'єкт анонімного класу, успадкованого від Handler, який передає опрацювання повідомлення методу handleIPDetermination Message() класу головної активності.

Обробник кнопки "Determine IP address" має такий вигляд, як показано на рис. 6.6.

```
        // …
            public void onDetermineIPAddressClick(View v)\
            ipTextView.setText("");
    progressBar.setVisibility(View.VISIBLE);
            Thread determinationThread = New Thread()
              @Override
              public void run()
                determineIPAddress();
            }
      };
             determinationThread.start();
      };
    // …
```

Рис. 6.6. **Обробник кнопки Determine IP address**

Цей обробник очищає поле виведення результату, робить видимим індикатор прогресу, а потім ініціює виконання методу determineIPAddress() класу головної активності на окремому потоці виконання. Зазначений метод здійснює запит до вебсервісу, опрацьовує повернене значення у форматі JSON та відправляє визначену IP-адресу на головний потік виконання у вигляді повідомлення (рис. 6.7).

```
// …
  private void determineIPAddress()
    try
        URL url = new URL(
"http://ip2country.sourceforge.net/ip2c.php?format=JSON");
        HttpURLConnection conn = (HttpURLConnection)
url.openConnection(); conn.connect();
        BufferedReader reader = new BufferedReader(
          new InputStreamReader(conn.getInputStream()));
        String ip = (String) new JSONObject(reader.readLine()).get("ip");
reader.close();
        handler.sendMessage(handler.obtainMessage(0, 0, 0, ip));
         cat (Exception e)
        handler.sendMessage(handler.obtainMessage(0, 0, 0,
e.getMessage()));
      }
// …}
```

Рис. 6.7. **Опрацювання запиту до вебсервісу**

Клас Handler опрацьовує повідомлення, викликаючи метод handleIP DeterminationMessage() і передаючи йому повідомлення як аргумент.

Цей метод приховує індикатор прогресу, вилучає певну IP-адресу з повідомлення та поміщає її в текстове поле.

Крім використання класу Handler, Android API надає простіший підхід для організації асинхронного виконання. Цей спосіб засновано на застосуванні класу AsyncTask, що інкапсулює деяке завдання, що має вхідні та вихідні параметри й потребує виконання на окремому потоці. Під час використання цього класу немає необхідності створювати потік явно, достатньо лише успадкувати від AsyncTask власний клас та перевизначити кілька методів.

AsyncTask – це параметризований клас. Його параметри позначають <Params, Progress, Result> і вони є типами вхідних даних, проміжного й остаточного результату завдання. Основні перевизначені методи класу містять:

```
protected void onPreExecute();
protected Result doInBackground(Params… params);
protected void onProgressUpdate(Progress… values);
protected void  onPostExecute(Result result);
```

Головний із перелічених методів — doInBakground(). Він містить власне код завдання, яке виконують на окремому потоці. Інші наведені методи викликають на головному потоці виконання й, отже, можуть звертатися до елементів інтерфейсу користувача. Синхронізацію між потоками здійснює клас AsyncTask автоматично. Методи onPreExecute() та onPostExecute() виконують, відповідно, перед запуском та після завершення асинхронного завдання. Виконання методу onProgressUpdate() ініціює виклик Protected void publishProgress(Progress… values); із коду методу doInBakground(). Цю можливість використовують для виведення користувачу проміжних результатів або статусу під час виконання завдання.

6.4. Провайдери контенту

Провайдери контенту входять до основних компонентів Android-застосунків разом з активностями та сервісами. Їхнє завдання полягає в наданні даних безлічі застосунків без прив'язування до конкретного способу зберігання цих даних.

Операції, що надає провайдер контенту, є аналогічними типовим операціям роботи з базою даних і містять додавання, оновлення та видалення записів, а також виконання запитів на вибірку даних. Слід зазначити, що природа цих даних може бути довільною, оскільки і їхня структура, і спосіб збереження повністю перебувають у зоні відповідальності провайдера контенту.

Для взаємодії із провайдером контенту клієнти специфікують URI даних, над якими виконують дію, як-от: *content://назва_провайдера_контент/специфікація_даних*. Назва провайдера контенту має збігатися зі значенням,

яке вказують під час реєстрації провайдера контенту у файлі маніфесту. Специфікація даних може мати будь-який формат, однак загальноприйнятою є REST-подібна специфікація.

Прикладом може бути стандартний провайдер контенту, що надає доступ до контактів користувача мобільного пристрою. Цей провайдер використовує URI типу content://com.android.contacts/contacts для визначення всіх доступних контактів та URI *content://com.android.contacts/contacts/1* для доступу до даних контакту під номером 1. Причому останній URI можна використовувати не тільки для визначення даних контакту, але й для їхньої зміни або видалення.

Метод insert() підтримує лише URI виду content://ru.ac.uniyar.todoslist.contentprovider/todos, оскільки в момент додавання жодного ідентифікатора запису ще не надано. Метод delete() для цього провайдера контенту не реалізовано (викидає виняток), оскільки у прикладі, що розглядають, необхідності в ньому немає.

Розглядаючи реалізацію перелічених методів, можна зазначити, що основна функція провайдера контенту в цьому разі ведеться до перетворення запитів до провайдера на запити до БД, що зберігає дані про завдання.

Кожен провайдер контенту має бути зареєстрованим у файлі маніфесту. У нашому прикладі реєстрація має такий вигляд (рис. 6.8).

```
<provider
    android:name=".ToDoContentProvider"
    android:authorities="ru.ac.uniyar.todoslist.contentprovider" >
</provider>.
```

Рис. 6.8. **Провайдер контенту, зареєстрований у файлі маніфесту**

Після встановлення програми доступ до провайдера контенту можна буде здобути з будь-яких застосунків, використовуючи відповідні URI.

Під час використання провайдерів контенту є дуже корисною можливість асинхронного завантаження даних за допомогою класу CursorLoader. Цей клас здійснює запит до провайдера контенту на окремому потоці

виконання та викликає callback-метод, коли завантаження завершується і можна використовувати визначені дані.

Для використання класу CursorLoader необхідний клас, який реалізує інтерфейс LoaderManager.LoaderCallbaks, що визначає всі необхідні callback-методи. За згодою таким класом зазвичай є клас активності (рис. 6.9).

```
public class MainActivity extends Activity
implements LoaderManager.LoaderCallbacks<Cursor>
private SimpleCursorAdapter adapter;
@Override
public void onCreate(Bundle savedInstanceState) super.onCreate(savedInstanceState);
setContentView(R.layout.main);
ListView todoListView = (ListView) findViewById(R.id.todoList);
todoListView.setOnItemClickListener(
new ListView.OnItemClickListener()
@Override
public void onItemClick(AdapterView<?> parent, View view,
int position, long id) onToDoListItemClick(id);
                    }
 getLoaderManager().initLoader(0, null, this);
         String[] from = new String[]  { "title", "description"}
         int[] to = new int[]     R.id.titleText,    R.id.descriptionText   ;
             adapter = new SimpleCursorAdapter(this, R.layout.todo item, null, from, to, 0
             todoListView.setAdapter(adapter);
          }
      //
```

Рис. 6.9. **Код класу активності**

Зауважмо, що, на відміну від методу onCreate(), у цьому прикладі немає запиту до БД, а об'єкт класу SimpleCursorAdapter створюють не пов'язаним із конкретним курсором (третій параметр конструктора дорівнює null). Це відбувається тому, що курсор буде створено динамічно класом CursorLoader після завантаження даних. Метод initLoader() ініціює запуск методу onCreateLoader() інтерфейсу LoaderManager.LoaderCallbaks (рис. 6.10).

```
// …
  @Override
  public Loader<Cursor> onCreateLoader(int id, Bundle args)
    return new CursorLoader(this,
     Uri.parse("content://ru.ac.uniyar.todoslist.contentprovider/todos"),
    null, null, null, null);
// …}
```

Рис. 6.10. **Метод onCreateLoader() інтерфейсу LoaderManager.LoaderCallbaks**

Після закінчення завантаження даних викликають callback-метод onLoadFinished(), якому передають курсор, пов'язаний із визначеними даними (рис. 6.11).

```
  @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor data)
    adapter.swapCursor(data);
// …}
```

Рис. 6.11. **Callback-метод**

Метод swapCursor() призводить до відображення даних у списку головної активності. Коли дані стають недоступними, викликають метод onLoaderReset (рис. 6.12).

```
// …
  @Override
    public void onLoaderReset(Loader<Cursor> loader)
     adapter.swapCursor(null);
// …}
```

Рис. 6.12. **Метод onLoaderReset**

Під час використання провайдера контенту вставлення й оновлення даних відбуваються практично так само, як і в разі безпосереднього виконання операцій із базою даних. Єдина відмінність полягає в тому, що методи визначення та зміни даних викликають не на об'єкті класу SQLiteDatabase, а на об'єкті класу ContentResolver, що надає доступ до даних через провайдер контенту, ідентифікованого за URI (рис. 6.13).

```
    // …
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
if (resultCode != RESULT OK) return; ContentValues cv = new ContentValues();
cv.put("title", data.getStringExtra("title")); cv.put("description",

data.getStringExtra("description")); cv.put("dueDate",
data.getStringExtra("dueDate"));
if (data.hasExtra("id"))
getContentResolver().update(
Uri.parse("content://ru.ac.uniyar.todoslist.contentprovider/todos/" + data.getIntExtra("id",
0)), cv, null, null);
            else
              getContentResolver().insert(
             Uri.parse("content://ru.ac.uniyar.todoslist.contentprovider/todos/"), cv);
      }}
      public void onToDoListItem Click(long id)
        Cursor todoCursor = getContentResolver().query(
            Uri.parse("content://ru.ac.uniyar.todoslist.contentprovider/todos/" + id), null, null,
            null, null);
        todoCursor.moveToNext();
        Intent intent = new Intent(this, ToDoEditorActivity.class); intent.putExtra("id",
        todoCursor.getInt(
            todoCursor.getColumnIndex(" id"))); intent.putExtra("title",
        todoCursor.getString(
            todoCursor.getColumnIndex("title"))); intent.putExtra("description",
        todoCursor.getString(
            todoCursor.getColumnIndex("description")));
        intent.putExtra("dueDate", todoCursor.getString(
            todoCursor.getColumnIndex("dueDate")));
            startActivityForResult(intent, 1);
      }
    } // class MainActivity
```

Рис. 6.13. **Опис класу ContentResolver**

Важлива особливість класу CursorLoader полягає в тому, що він автоматично відстежує зміни даних, визначення яких він здійснює. Під час зміни цих даних процес завантаження відбувається знову без додаткових зусиль із боку програміста. Тому під час зміни даних немає необхідності у виклику методу requery().

Практична складова до підрозділу 6
Зміна орієнтації екрана смартфона

Є два режими орієнтації екрану смартфона – портретний та альбомний. Під час повороту екрана телефона з одного режиму в інший застосунок має коректно перемальовувати інтерфейс. Створюймо новий проєкт на основі Hello, World!. В **activity_main.xml** у текстовому режимі змінюймо розмітку **RelativeLayout** на **LinearLayout**, потім у властивостях розмітки **orientation** на **vertical**. У доповненні до наявного елементу **TextView** додаймо шість кнопок **Button**. Назви кнопок установлюймо через ресурси (рис. 6.14).

```
<?xml version ="1.0" encoding ="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools" android:id ="@ + id /
        activity_main" android:layout_width ="match_parent" android:layout_height
        ="match_parent"
       android:paddingBottom ="@ dimen / activity_vertical_margin" android:paddingLeft
="@ dimen / activity_horizontal_margin" android:paddingRight ="@ dimen /
activity_horizontal_margin" android:paddingTop ="@ dimen / activity_vertical_margin"
tools:context ="ua.edu.uipa.ikpt.lab3.MainActivity" android:orientation ="vertical">
        <TextView
                    android:layout_width ="match_parent"
              android:layout_height ="wrap_content" android:text
              ="Hello World!" android:id ="@ + id / textView" />
        <Button
                    android:text ="@ string / one"
              android:layout_width ="match_parent"
              android:layout_height ="wrap_content" android:id
              ="@ + id / button1" />
        <Button
                    android:text ="@ string / two"
              android:layout_width ="match_parent"
              android:layout_height ="wrap_content" android:id
              ="@ + id / button2" />
        <Button
                    android:text ="@ string / three"
              android:layout_width ="match_parent"
              android:layout_height ="wrap_content" android:id =
              "@ + id / button3" />
```

Рис. 6.14. **Зміни у файлі activity_main.xml**

```
<Button
                android:text ="@ string / four"
        android:layout_width ="match_parent"
        android:layout_height ="wrap_content" android:id
        ="@ + id / button4" />
<Button
```

Закінчення рис. 6.14

Переглядаймо у Preview і запускаймо в емуляторі. У портретному режимі все відображається нормально, але якщо змінити орієнтацію на альбомний формат виходить непорядок: не всі кнопки поміщаються (рис. 6.15).

Рис. 6.15. **Результати перегляду в режимі Preview**

Щоб уникнути такої проблеми, необхідно в альбомному режимі по-іншому скомпонувати кнопки. Наприклад, розташувати їх не підряд один за одним, а розподілити на пари.

Перейдіть у режим дизайну вашої основної розмітки. Нагорі на панелі інструментів є значок **Layout Variants** (найбільш правий значок). Клацніть на ньому та виберіть пункт **Create LandscapeVariation**. З'явиться готовий файл у папці **res/layout-land**. Уміст файлу з портретної орієнтації скопійовано в інший файл і його можна редагувати. Надалі вибирати

файл розмітки можна, вибираючи **Switch to layout-land**, або **Switch to layout**, або відповідні файли (рис. 6.16).

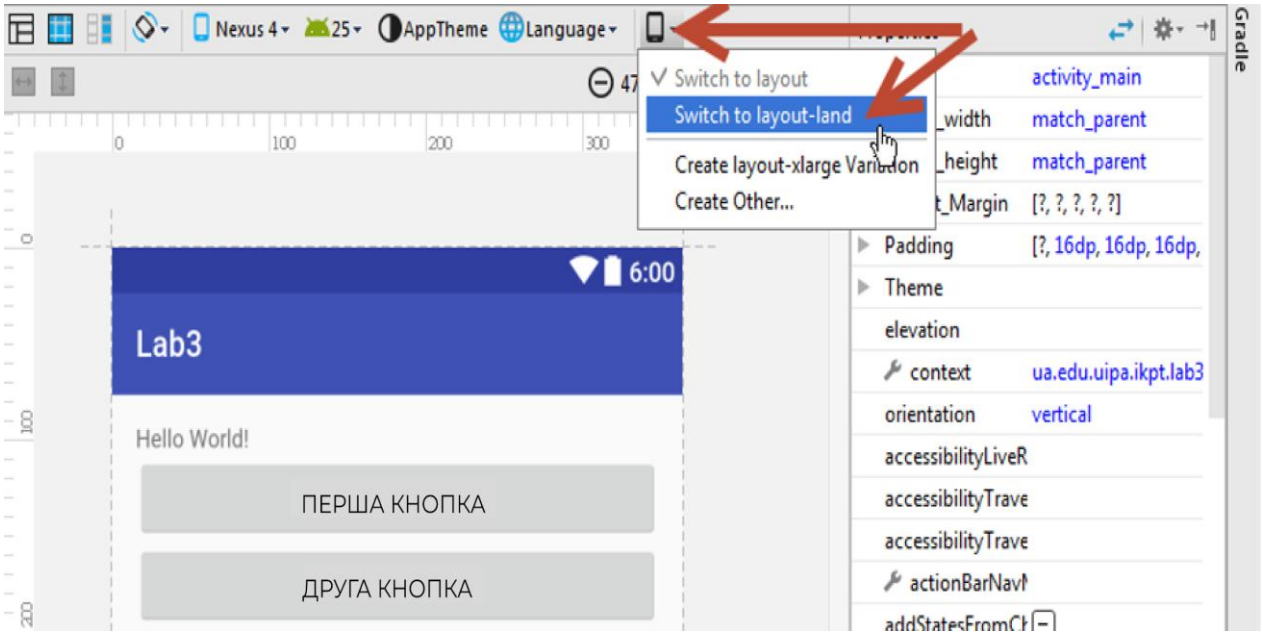


Рис. 6.16. **Вибір файлу розмітки**

Слід пояснити, як було створено іншу розмітку. Разом із папкою **layout** було створено другу папку **layout-land** і в неї скопійовано файл **activity_main_xml**. У режимі перегляду файлів **Android** відображаються файли **activity_main_xml** і **activity_main_xm l** (land). Насправді це два файли з однаковими назвами, але в різних папках, у цьому можна переконатися, переключившись у режим Project (рис. 6.17).

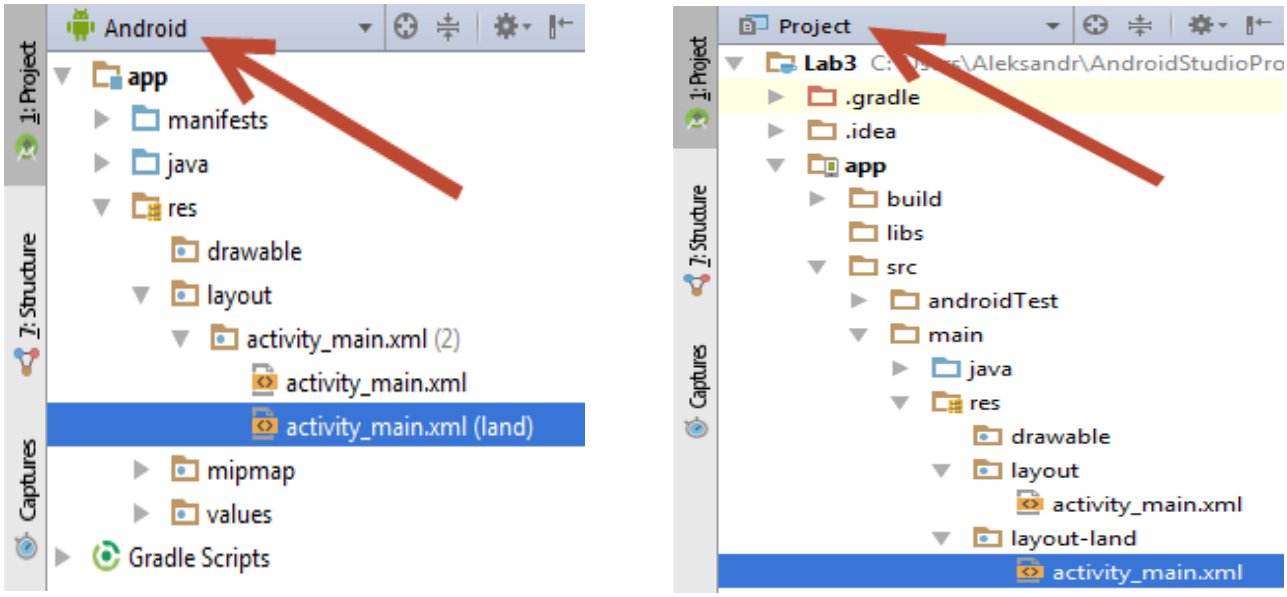


Рис. 6.17. **Перегляд у режимі Project**

Скористаймося контейнером **TableLayout**. Із його допомогою можна зробити кнопки на дві колонки й помістити їх у три рядки.

Відкриймо створений файл і модифікуймо його. Зверніть увагу на рядок **android:stretchColumns ="*"**, він центрує кнопки (рис. 6.18).

```
<TextView
        android:layout_width ="match_parent"
        android:layout_height ="wrap_content"
        android:text ="Hello World!" android:id ="@ + id
        / textView"/>
<TableLayout
        android:layout_width ="match_parent"
        android:layout_height ="wrap_content"
        android:stretchColumns ="*" android:id ="@ + id

        /tableLayout">
        <TableRow>
                  <Button
                android:text ="@ string / one"
                android:layout_width ="match_parent"
                android:layout_height ="wrap_content"
                android:id ="@ + id / button1" />
                  <Button
                android:text ="@ string / two"
                android:layout_width ="match_parent"
                android:layout_height ="wrap_content"
                android:id ="@ + id / button2" />
        </TableRow>
        <TableRow>
                  <Button
                android:text ="@ string / three"
                android:layout_width ="match_parent"
                android:layout_height ="wrap_content"
                android:id ="@ + id / button3" />
                  <Button
                android:text ="@ string / four"
                android:layout_width ="match_parent"
                android:layout_height ="wrap_content"
                android:id ="@ + id / button4" />
```

Рис. 6.18. **Модифікація створеного файлу**

```
</TableRow>
<TableRow>

        android:text ="@ string / five"
        android:layout_width ="match_parent"
        android:layout_height ="wrap_content"
        android:id ="@ + id / button5" />

        <Button

        android:text ="@ string / six"
        android:layout_width ="match_parent"
        android:layout_height ="wrap_content"
        android:id ="@ + id / button6" />

</TableRow>
```

Закінчення рис. 6.18

Запускаймо програму та перевіряймо. Дуже добре, тепер видно всі кнопки (рис. 6.19).

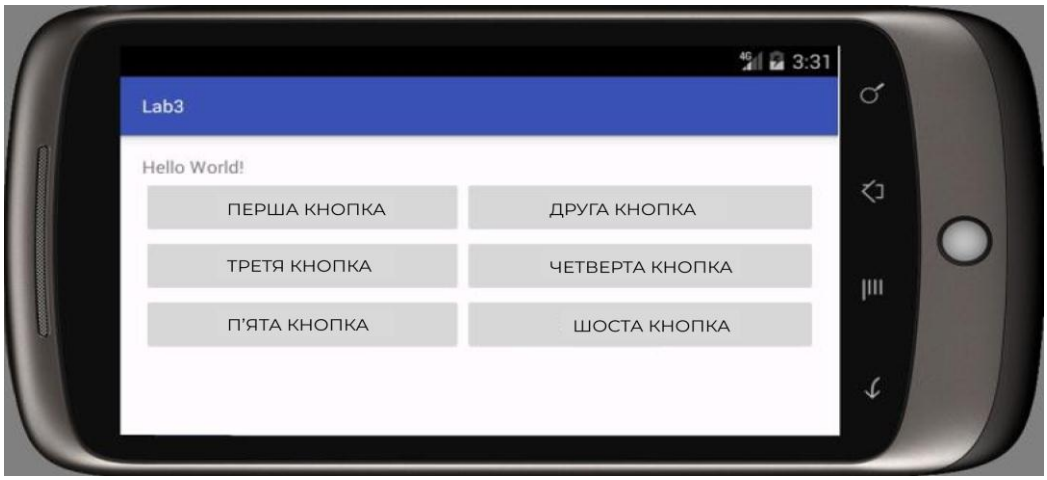


Рис. 6.19. **Результати запуску програми**

Коли створюєте альтернативну розмітку, то не забувайте додавати всі компоненти, до яких будете звертатися програмно, інакше буде помилка. Припустімо, ви забули додати шосту кнопку. У портретному режимі програма буде працювати, а коли користувач переверне екран, то активність буде форматувати всі компоненти для роботи, а кнопки-то й немає крах програми.

Часто програма має виконувати різні дії, залежно від орієнтації екрана. Додаймо ще одну кнопку, під час натискання на яку в **TextView** буде виводитися повідомлення: **Альбомна орієнтація** або **Портретна орієнтація**.

Щоб із коду дізнатися поточну орієнтацію, можна створити таку функцію, додавши її в MainActivity.java перед закриття фігурної дужки. У прикладі на рис. 6.20 використовували дві поширені системні константи для орієнтації.

```
private String getScreenOrientation () {
        if (getResources (). getConfiguration ().orientation ==
Configuration.ORIENTATION_PORTRAIT)
                        return "Портретнаорієнтація";
        else if (getResources (). getConfiguration ().orientation ==
Configuration.ORIENTATION_LANDSCAPE)
                        return "Альбомнаорієнтація";
                else
                        return "";
}
```

Рис. 6.20. **Використання системних констант для орієнтації**

Щоб змінювати текст у текстове поле, перед **onCreate** слід оголосити змінну (рис. 6.21).

```
private TextView textView;
```

Рис. 6.21. **Оголошення змінної**

А в **onCreate** цю змінну слід ініціалізувати (рис. 6.22).

```
textView = (TextView) findViewById (R.id.textView);
```

Рис. 6.22. **Ініціалізація змінної**

Процедуру опрацювання клацання кнопки для перевірки орієнтації в activity_main.xml назвімо **onCheckButtonClick.** Не забуваймо додати

onCheckButtonClick в **обидва варіанти** xml-файлів розмітки екрана (рис. 6.23).

```
public void onCheckButtonClick (View view) {
                    textView.setText (getScreenOrientation ());
```

Рис. 6.23. **Процедура опрацювання клацання кнопки для перевірки орієнтації в activity_main.xml**

У результаті можна програмно визначати орієнтацію.

Можна визначити поточну орієнтацію, але в який бік повернули пристрій, залишається невідомим. Адже його можна повернути уліво, управо або взагалі догори низом. Напишімо іншу функцію (рис. 6.24).

```
private String getRotateOrientatin () {
    int rotate = getWindowManager (). getDefaultDisplay (). getRotation ();
        switch (rotate) {
                case Surface.ROTATION_0:
                    return "Чи
неповертали"; case
Surface.ROTATION_90:
                    return "Повернули
протигодинникової"; case Surface.ROTATION_180:
                    return "Повернули на
180градусів"; case Surface.ROTATION_270:
                    return "Повернули
загодинниковою"; default:
                        return "Незрозуміло";
```

Рис. 6.24. **Функція визначення повороту**

Із поворотом екрана виникає одна дуже неприємна проблема. Замисліться над значенням слів, що під час повороту екрана активність створюється заново. Це означає, що змінні знову набувають значення 0, тобто скидаються в початкове значення.

Створюймо змінну **mCount** (до onCreate, рис. 6.25).

```
private int mCount=0;
```

Рис. 6.25. **Створення змінної mCount**

Модифікуймо функцію. Під час натискання на кнопку значення змінної **mCount** спочатку збільшується на одиницю, а потім дописується на початку тексту (рис. 6.26).

```
public void onCheckButtonClick (View view) {
        textView.setText (++mCount + "" + getRotateOrientatin ());
```

Рис. 6.26. **Модифікація функції**

Кожне натискання збільшує значення, але тільки до тих пір, поки не повернуто екран. Після повороту рахунок починають спочатку.

Для збереження значень змінних цілей в активності є спеціальний метод **onSaveInstanceState**(), який викликає система перед методами **onPause**(), **onStop ()** і **onDestroy**(). Метод дозволяє зберегти значення простих типів в об'єкті **Bundle**. Клас **Bundle** – це простий спосіб зберігання даних ключ/значення.

Створімо ключ із назвою **KEY_COUNT**. В Android Studio є живі шаблони, що дозволяють швидко створити ключ. Уведіть до методу **onCreate ()** малими літерами слово **key**, під час набору з'явиться підказка. Натискаймо **Enter** і маємо заготовку. Після символу підкреслення вводьмо назву ключа. У результаті маємо ключ наступного виду, як показано на рис. 6.27.

```
private static final String KEY_COUNT = "COUNT";
```

Рис. 6.27. **Ключ до методу onCreate ()**

Далі створюймо метод **onSaveInstanceState ()** після методу **onCreate**(). Під час набору назви методу підказка покаже, що є два

методи. Вибираймо метод з одним із параметрів (зазвичай він іде другим). Записуймо у ключ значення лічильника (рис. 6.28).

```
protected void onSaveInstanceState (Bundle outState) {
super.onSaveInstanceState (outState); outState.putInt(KEY_COUNT, mCount);
}
```

Рис. 6.28. **Створення методу onSaveInstanceState ()**

А в методі і**onCreate ()** робімо невелику перевірку та відновлюймо значення mCount (рис. 6.29).

```
protected void onCreate (Bundle savedInstanceState) {
super.onCreate (savedInstanceState); setContentView
(R.layout.activity_main);
textView= (TextView) findViewById (R.id.textView);
if (savedInstanceState! = null){
mCount= savedInstanceState.getInt(KEY_COUNT, 0);
textView.setText(mCount + "" + getRotateOrientatin ());

} }
```

Рис. 6.29. **Відновлення значення mCount**

У методі параметра міститься об'єкт **Bundle**. Тут його названо **savedInstanceState**.

Назви можна придумувати самим. Головне, що об'єкт містить збережене значення змінної під час повороту. Під час першої активації об'єкта немає **(null)**, а потім його створено кодом. Для цього й потрібна перевірка. Зверніть увагу, що тут одиницю до лічильника не додаємо, як у кнопці. Якщо скопіювати код у кнопки, то вийде, що лічильник буде збільшуватися самостійно під час поворотів без натискання на кнопку. Але це може ввести в оману користувача.

Цей спосіб використовують для збереження проміжних результатів під час дії програми.

У результаті маємо вигляд, показаний на рис. 6.30.

Рис. 6.30. **Результат виконання роботи з повороту**

На рис. 6.31 наведено весь код у MainActivity.java.

```
public class MainActivity extends AppCompatActivity {

    private TextView textView;
   private int mCount=0;
    private static final String KEY_COUNT = "COUNT";
    @Override
    protected void onCreate (Bundle savedInstanceState) {
          super.onCreate (savedInstanceState); setContentView
          (R.layout.activity_main); textView= (TextView) findViewById
          (R.id.textView); if (savedInstanceState! = null){
```

Рис. 6.31. **Код в MainActivity.java**

```
                    mCount= savedInstanceState.getInt(KEY_COUNT, 0);
                    textView.setText(mCount + "" + getRotateOrientatin
());
}
  // setRequestedOrientation
  (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE); // для альбомного
  режиму
            // або
  //setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRA IT); // для
  портретного режиму
        }
        @Override
        protected void onSaveInstanceState (Bundle outState) {
            super.onSaveInstanceState (outState);
            outState.putInt(KEY_COUNT, mCount);
        }
        public void onCheckButtonClick (View view) {
            textView.setText (++mCount + "" + getRotateOrientatin
());
        }
        private String getScreenOrientation () {
            if (getResources (). getConfiguration ().orientation ==
Configuration.ORIENTATION_PORTRAIT)
                return "Портретна орієнтація";
            else if (getResources (). getConfiguration ().orientation ==
  Configuration.ORIENTATION_LANDSCAPE)
                return "Альбомна орієнтація";
            else
                return "";
        }
        private String getRotateOrientatin () {
            int rotate = getWindowManager (). getDefaultDisplay (). getRotation ();
            switch (rotate) {
                case Surface.ROTATION_0:
                    return "Чи не повертали";
                case
                    Surface.ROTATION_90:
                    return "Повернули проти годинникової"; case
                Surface.ROTATION_180:
                    return "Повернули на 180 градусів"; case
                Surface.ROTATION_270:
                    return "Повернули за годинниковою"; default:
                    return "Незрозуміло";
 }}}
```

Закінчення рис. 6.31

Указаний код дозволяє зберігати проміжні результати під час дії програми.

Контрольні запитання для самоперевірки

1. Що таке "асинхронне виконання"? У яких випадках його використовують? Які завдання воно вирішує?

2. Перелічіть засоби асинхронного виконання, що надають Android API.

3. Що таке "черга повідомлень"? Яку функцію виконує клас Handler та як його правильно використовувати?

4. Навіщо призначено клас AsyncTask? Як його використати?

5. Поясніть, що таке "синхронізація потоків". У яких випадках вона є потрібною? Як засоби платформи Android допомагають вирішувати завдання синхронізації?

6. Модифікуйте приклади із цієї теми таким способом, щоб асинхронне виконання не припинялося під час зміни конфігурації програми (наприклад, зміни орієнтації екрана).

7. Що таке "провайдер контенту"? Яку роль відіграють постачальники контенту в інфраструктурі Android?

8. Для чого потрібен URI для використання провайдера контенту? Із яких частин він складається? За якими правилами формують цей URI?

9. Які методи класу ContentProvider необхідно перевизначати під час реалізації провайдера контенту?

10. Яким є призначення класу CursorLoader? Які переваги має використання класу CursorLoader, порівняно з безпосереднім виконанням запитів до провайдера контенту на головному потоці програми?

11. Навіщо призначено callback-методи інтерфейсу LoaderManager.LoaderCallbaks? Наведіть приклад їхнього використання.

Рекомендована література: [1 – 3; 5 – 7; 9 – 13; 15; 17; 18].

Використана та рекомендована література

1. Власій О. О. Розробка мобільних додатків засобами блочного програмування : навч.-метод. посіб. / О. О. Власій, М. Д. Винничук. – Івано-Франківськ : Прикарпатський національний університет імені Василя Стефаника, 2021. – 130 с.

2. Давидов М. В. Програмне забезпечення мобільних пристроїв : навч. посіб. / М. В. Давидов, А. Б. Демчук, О. В. Лозинська. – Львів : Вид-во "Новий Світ-2000", 2020. – 218 с.

3. Павлиш В. А. Основи інформаційних технологій і систем : навч. посіб. / В. А. Павлиш, Л. К. Гліненко. – Львів : Вид-во Львівської політехніки, 2021. – 500 с.

4. Розробка мобільних застосунків для OS Android : навч. посіб. / М. Л. Дворецький, Ю. О. Нездолій, С. В. Дворецька, І. О. Кандиба. – Миколаїв : вид-во ЧНУ ім. Петра Могили, 2021. – 140 с.

5. Aralova N. I. The Method of Technology Evaluation Based on Improved Cost Approach / N. I. Aralova, O. Y. Kyiashko // Science and Innovation. – 2017. – No 13 (3). – P. 65–76.

6. Hrabovskyi Y. Development of Information visualization methods for use in multimedia applications / Y. Hrabovskyi, N. Brynza, O. Vilkhivska // EUREKA: Physics and Engineering. – 2020. – No 1. – P. 3–17.

7. Hrabovskyi Y. Development of the optimization model of the interface of multimedia edition / Y. Hrabovskyi, V. Fedorchenko // EUREKA: Physics and Engineering. – 2019. – No 3. – P. 3–12.

8. Hrabovskyi Y. Methods of Developing the Event-agency Site / Y. Hrabovskyi // Збірник наукових праць Харківського національного університету Повітряних Сил. – 2021. – Вип. 4 (70). – С. 70–76.

9. Hrabovskyi Y. The methodology of developing a mobile application design for creating a genealogical tree / Y. Hrabovskyi, Yu. Brusiltseva // Поліграфія і видавнича справа. – 2022. – No 1 (83). – С. 66–78.

10. Hryshchuk R. Synergetic control of social networking services actors' interactions / R. Hryshchuk // Recent Advances in Systems, Control and Information Technology. – 2017. – No 543. – P. 34–42.

11. Khamula O. H. Factors of influence of interface use based on mobile applications / O. H. Khamula, N. V. Soroka, S. P. Vasiuta // Наукові записки [Української академії друкарства]. – 2016. – № 2. – С. 28–36.

12. Safonov I. Adaptive Image Processing Algorithms for Printing / I. Safonov // Singapore : Springer. – 2018. – 304 p.

13. Бучач А. Кодуємо для Android [Електронний ресурс] / А. Бучач. – Режим доступу : http://bit.ly/2UFrvPM.

14. Засоби моніторингу та аналізу мережі [Електронний ресурс]. – Режим доступу : https://www.arc-it.net/html/archuse/archuse.html.

15. Цирульник С. М. MIT App Inventor: створення android-додатку лабораторного практикуму без програмування [Електронний ресурс] / С. М. Цирульник // Відкрите освітнє середовище сучасного університету. – 2018. – Вип. 4 – С. 91–95. – Режим доступу : http://nbuv.gov.ua/UJRN/oeeemu_2018_4_12.

16. Developing Android Apps with App Inventor / Coursera : [вебсайт]. – Access mode : https://www.coursera.org/learn/app-inventor-android.

17. Meet and Code : [вебсайт] "Дівчата програмують Android додатки" : Онлайн-курс. – Режим доступу : https://meet-and-code.org/be/nl/event-show/4576.

18. Teaching with App Inventor. MIT App Inventor : [вебсайт]. – Access mode : http://appinventor.mit.edu/explore/teach.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Пушкар Олександр Іванович
Грабовський Євген Миколайович

Проєктування додатків для мобільних пристроїв

Навчальний посібник

Відповідальний за видання *О. І. Пушкар*

Відповідальний редактор *О. С. Вяткіна*

Редактор *О. Г. Доценко*

Коректор *О. Г. Доценко*

План 2023 р. Поз. № 7-ЕНП. Обсяг 167 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***