МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти	Перший (бакалаврський)
Спеціальність	Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення
Група	6.04.121.010.18.1

ДИПЛОМНИЙ ПРОЄКТ

на тему: «Розроблення мобільного застосунку для здійснення регулярних платежів»

Виконав: студент Данило ТИМОШЕВСЬКИЙ

Керівник: к.т.н., проф. Олександр ЩЕРБАКОВ

Рецензент:
к.т.н., доцент кафедри інформатики
та комп'ютерної техніки ХНЕУ ім.С.Кузнеця
Наталя БРИНЗА

Харків – 2022 рік

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту: 73 с., 38 рис., 21 табл., 2 додатка, 27 джерел.

Об’єктом проєктування є функціональні елементи, архітектура, інформаційне і програмне забезпечення модуля здійснення регулярних платежів.

Мета дипломного проєкту – створення зручного мобільного застосунку для управління існуючими та здійснення нових регулярних платежів на базі клієнт-серверної архітектури [1], Web- та мобільних технологій.

Предмет дослідження – методи та технології здійснення регулярних платежів та управління ними шляхом використання мобільних та Web-технологій.

Пояснювальна записка дипломного проєкту містить результати проєктування та реалізацію модуля здійснення регулярних платежів на базі мобільних та web-технологій.

Метод проєктування – використання сучасних спеціалізованих програмних засобів та систем, таких як Visual Paradigm та Lucidchart. Модуль розроблений у програмному середовищі Android Studio та PyCharm.

Отриманий модуль реалізує здійснення регулярних платежів за допомогою web-технологій Django та мобільних технологій Android API, що є найбільш стабільними і розповсюдженими серед наявних технологій відповідних сфер.

Результат розробки буде впроваджений на підприємстві ФОП Тимошевський Данило Сергійович.

DJANGO, ANDROID API, CASE-ДІАГРАМИ, WEB-ТЕХНОЛОГІЇ, МОБІЛЬНІ ТЕХНОЛОГІЇ, БАЗА ДАНИХ, РЕГУЛЯРНІ ПЛАТЕЖІ, ПІДПИСКИ, ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

ABSTRACT

The bachelor's thesis report: 73 pages, 38 figures, 21 tables, 2 appendices, 27 sources.

The functional elements, architecture, information and software of the subscription managing module are the objects of design.

The purpose of the diploma project is to create a convenient mobile application for managing existing and creating new regular payments based on client-server architecture, Web and mobile technologies.

Subject of research – methods and technology of making regular payments and managing them by using mobile and Web technologies.

The bachelor's thesis report contains the results of the design and implementation of the subscription managing module based on mobile and Web technologies.

The method of design is the use of modern specialized software and systems like Visual Paradigm and Lucidchart. The module was developed in the Android Studio and PyCharm software environments.

The resulting module implements the managing of regular payments using Web-technologies, namely Django and mobile technologies, namely Android API, which are one of the most stable and popular solutions among the existing technologies in corresponding spheres.

The result of the development will be implemented at "FOP Tymoshevskyi Danylo Serhiiovych" enterprise.

DJANGO, ANDROID API, CASE DIAGRAMS, WEB TECHNOLOGIES, MOBILE TECHNOLOGIES, DATABASE, REGULAR PAYMENTS, RECURRING PAYMENTS, SUBSCRIPTIONS, OBJECT-ORIENTED PROGRAMMING.

ЗМІСТ

ВСТУП

Наразі у світі зберігається тенденція переходу сфери електронних послуг та розваг з моделі одноразової покупки на модель регулярної підписки, що дозволяє продовжити життєвий цикл, а також термін підтримки та вдосконалення таких сервісів.

Також, з настанням карантину, люди почали більше користуватись всесвітньо відомими сервісами у сфері розваг, абсолютна більшість яких заснована на моделі саме регулярної підписки (наприклад, Netflix, YouTube Premium та Xbox Game Pass).

До того ж, багато благодійних фондів в Україні постають перед дилемою: платити комісію стороннім сервісам (наприклад, фонд «Повернись живим» має опцію оформлення регулярних платежів через Patreon, який бере комісію у розмірі від 5 до 12 відсотків), або приймати одноразові пожертви, що може бути незручним для користувачів, які не мають заощаджень або не можуть власноруч відслідковувати бажані витрати, і при цьому хочуть, щоб усі гроші, що вони витрачають на благодійність, витрачались саме на неї.

На даний момент існує декілька застосунків, що дозволяють керувати регулярними платежами, але більшість з них дозволяє тільки нагадувати про спливання терміну оплати, а також не має автоматизованої генерації списку платежів (котрі у такому раз треба вводити вручну). Також жоден з таких застосунків не дозволяє тим чи іншим чином створювати такі регулярні платежі.

Тож, проблемою бізнесу, що підлягає вирішенню, є надання користувачу повного обсягу функціональності, що стосується перегляду, нагадування, управління та створення регулярних платежів у межах одного універсального застосунку, що дозволить користувачеві мати один інструмент для досягнення всіх описаних цілей, а не мати по окремому застосунку на функцію. Це скоротить час, що користувач витрачає на такі дії, а також підвищить зручність користування такими послугами та таким чином потенційно підвищить кількість коштів, що надходять різним благодійним фондам через те, що користувачі будуть менше сумніватись у потребі користування регулярними платежами, а також будуть знати, яка саме частка їх грошей буде перерахована на вказані ними рахунки.

Отже, метою дипломного проєкту є аналіз предметної області «Управління регулярними платежами» та розробка клієнт-серверного [1] застосунку для здійснення регулярних платежів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ «УПРАВЛІННЯ РЕГУЛЯРНИМИ ПЛАТЕЖАМИ»

1.1. Коротка характеристика об’єкту управління ТОВ «Аквелон Україна»

Akvelon було засновано в 2000 році висококваліфікованими IT-фахівцями, які мали великий досвід розробки складних програмних додатків. Унікальне поєднання знань у сфері IT та міцного фундаменту в загальних бізнес-операціях дозволяє Akvelon пропонувати високотехнологічні рішення, які використовуються нашими клієнтами по всьому світу.

Штаб-квартира компанії Akvelon розташована в Бельвю, штат Вашингтон.

Представництво «Аквелон» в Україні було відкрито в листопаді 2008 року і на даний момент налічує близько 300 осіб.

Як партнер і постачальник корпорації Microsoft, Аквелон бере безпосередню участь у розробці продуктів і внутрішніх проєктах Microsoft як в Україні, так і в США.

Першим бізнес-підрозділом «Аквелон Україна» був підрозділ .NET. Пізніше за ним пішли Java та мобільні рішення, такі як Android, iPhone/iPad, Blackberry, Windows Mobile, Symbian, BREW.

У 2010 році Akvelon став вендором одного з найбільших світових постачальників телекомунікаційних послуг T-Mobile USA, а також створив підрозділ дизайну [2].

У 2022 український підрозділ Аквелон має декілька офісів у м. Харків, а сама компанія має таких всесвітньо відомих партнерів, як Microsoft, Tideworks, T-Mobile, Reddit, GitHub, Nokia, at&t, Intel, Starbucks, Pinterest, Dropbox, LinkedIn, Hewlett Packard, Airbnb та багато інших [3].

Компанія забезпечує кар’єрний ріст своїм співробітникам, а також можливості навчання за програмою Trainee, де випускники або студенти, що не мають досвіду, можуть почати працювати з допомогою досвідчених менторів (інженерів програмного забезпечення, тестувальників, інженерів тестів та бізнес-аналітиків) на реальних проєктах у командах за технологією Agile/Scrum.

Також компанія дає можливість опанувати іншу професію у тій самій галузі за тією ж програмою, якщо співробітник має таке бажання (наприклад, з інженера програмного забезпечення стати бізнес-аналітиком).

Схема організаційної структури підприємства «Аквелон Україна» наведена на рис. 1.1.

Безпосередньо підприємством керують директор та виконавчий директор, причому директор напряму керує бухгалтерським обліком та юридичними справами, у той час як виконавчий директор займається більш широким рядом справ. По-перше, до його обов'язків входить керування фінансовою політикою підприємства, що включає стратегічні рішення з способів розрахунку компанії-замовника за виконану роботу на їх проєкті, а також способи виплати зарплатні робітникам. По-друге – до його компетентності входять рішення про відкриття нових офісів та модернізацію існуючих. По-третє – виконавчий директор також контролює поточні та підписує нові контракти з компаніями-замовниками (здійснює контроль над відділом аутсорсингу). І по-четверте, він контролює роботу інших відділів підприємства, до яких входять: відділ маркетингу, відділ людських ресурсів та відділ інформаційних технологій.

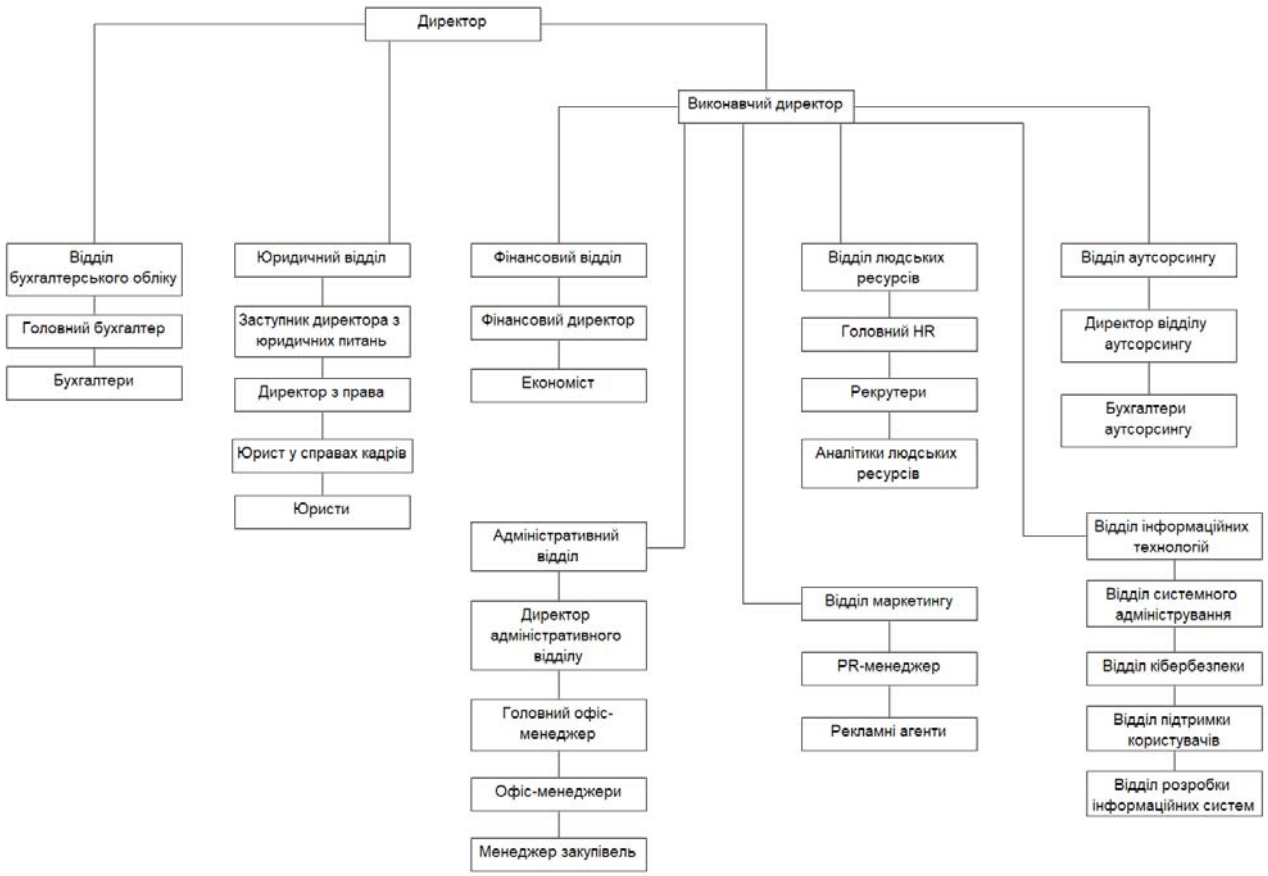

Рис. 1.1. Схема організаційної структури ТОВ «Аквелон Україна»

Наведені у схемі на рис. 1.1. відділи складають ядро функціональності підприємства, і мають кожен таке призначення:

- відділ бухгалтерського обліку забезпечує документообіг на підприємстві, а також керує особистою документацією замість співробітників (для тих співробітників, які бажають цього);
- юридичний відділ представляє підприємство у судах, регулює складання і підписання нових договорів від імені компанії, а також може надавати юридичні консультації та/або допомогу співробітникам у разі потреби;
- фінансовий відділ керує фінансами компанії, розподіляє прибутки, проводить оплату за виконані роботи та курує фінансові процедури під час виплат іноземних компаній-замовників підприємству;
- адміністративний відділ займається оснащенням офісів, закупівлями обладнання, контролем за безпекою на підприємстві та створенням комфортних умов праці для співробітників;
- відділ людських ресурсів займається пошуком нових працівників, аналізом ринку праці та забезпеченням комфорту співробітників;
- відділ маркетингу займається рекламою компанії для приваблення нових клієнтів, а також працівників;
- відділ аутсорсингу займається документообігом та контактами між підприємством та компаніями-клієнтами;
- відділ інформаційних технологій займається забезпеченням функціональності IT-інфраструктури компанії, захистом чутливої інформації, розробкою та підтримкою програмного забезпечення.

Проблемою бізнесу, яку слід вирішити, є відсутність універсального продукту для управління регулярними платежами. Бізнес-процесами [4], які потрібні для вирішення даної проблеми, є аналіз потреб клієнтів щодо функціональності такого застосунку, а також його розробка і впровадження.

Варто окремо більш детально розглянути найбільший відділ підприємства – відділ інформаційних технологій, який прямо пов’язаний з бізнес-процесом розробки програмного продукту, що дозволить вирішити проблему бізнесу. Організаційна структура цього відділу більш детально представлена на рис. 1.2.

Відділ інформаційних технологій підпорядковується виконавчому директору та складається з 4 основних відділів з чітко окресленою функціональністю кожного:

- відділ системного адміністрування підтримує у функціональному стані приватну мережу компанії і застосунки для багатьох цілей (збирання проєктів, координація робіт, документація, облік часу, автоматизоване тестування та ін.) у цій мережі, а також займається впровадженням нових застосунків за потреби, або розгортанням розроблюваних застосунків у замовника;

- відділ кібербезпеки займається постійним навчанням працівників тому, як себе поводити у випадках різних позаштатних ситуацій, пов’язаних з кіберзагрозами, як запобігати виникненню таких ситуацій; підтримує захист офісної мережі, апаратного та програмного забезпечення у ній від кіберзагроз, а також реагує на такі загрози і усуває їх;
- відділ підтримки користувачів займається підтримкою релізів продукту, збором інформації про можливі дефекти або побажання щодо покращення програмного продукту, що розробляється. Також формує своєрідний «міст» між замовниками та більш технічною частиною відділу інформаційних технологій – відділом розробки інформаційних систем – обробляючи інформацію, що надходить від користувачів, транслюють її у більш зрозумілу в контексті проєкту;
- відділ розробки інформаційних систем виконує аналіз, розробку і тестування продукту відповідно до потреб користувачів та інформації, яка надходить від користувачів програмним продуктом до відділу підтримки користувачів.

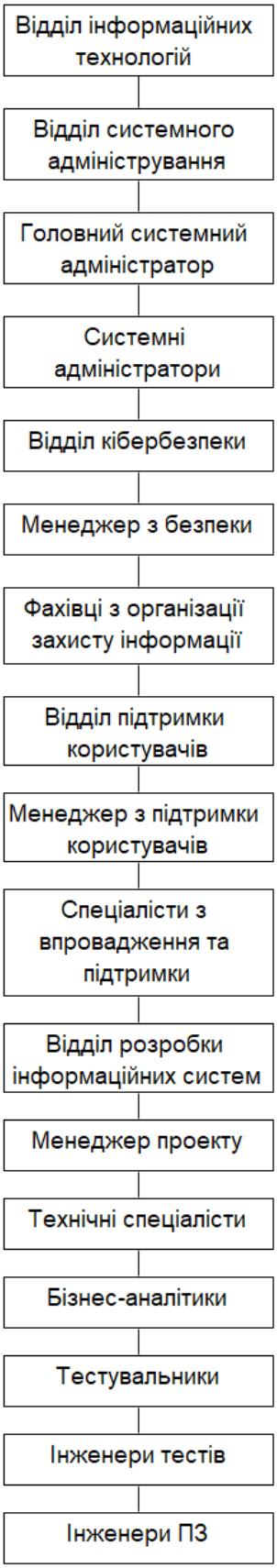

Рис. 1.2. Схема організаційної структури відділу інформаційних технологій

1.2. Опис предметної області «Управління регулярними платежами»

Процес управління регулярними платежами базується на користувацьких даних про банківські картки. Різні типи управління вимагають різного обсягу даних, котрі користувач має надати застосунку, і зазвичай від обсягу цих даних залежить рішення користувача про продовження або припинення користування застосунком. Тому під час розробки програмних продуктів, що оперують такою чутливою інформацією, як дані про рахунки з грошима, банківські картки і т.д., треба дозволяти користувачам максимум функціональності при мінімальній кількості специфічних даних.

Щоб охопити якомога більше користувачів, функціональність має бути чітко структурована за кількістю і типом необхідних для нормальної роботи даних.

Отже, маючи на увазі вищезазначене, процес управління регулярними платежами може містити такі найбільш розповсюджені функції:

- створення нових регулярних платежів (для цього від користувача потрібні дані платіжних засобів (банківських карток));
- додавання існуючих регулярних платежів для подальшого їх відстеження (у користувача є можливість зробити це вручну, без введення чутливих даних банківських карток);
- редагування інформації про додані вручну платежі (від користувача не треба додаткових даних);
- видалення інформації про додані вручну платежі (від користувача не треба додаткових даних);
- нагадування про близкість терміну списання коштів, що відносяться до регулярних платежів (від користувача не треба додаткових даних);
- підрахунок статистики витрат на додані вручну або новостворені платежі (від користувача не треба додаткових даних).

Для моделювання та графічного представлення процесу управління регулярними платежами було створено діаграму бізнес-процесу за допомогою CASE-засобу [5] Lucidchart у відповідності зі стандартом моделювання бізнес-процесів BPMN 2.0 [6]. Результат моделювання представлено на рис. 2.1.

Під час моделювання було обрано найпростіші сценарії функціональності цього процесу, на прикладі яких буде продемонстровано опис виконання процесу.

Представлений процес можна розбити на такі основні етапи:

- створення запиту. Користувач вирішує, яку саме дію він хоче виконати, і відправляє запит на цю дію за допомогою графічного інтерфейсу застосунку;

- обробка запиту. Застосунок отримує запит на дію від користувача. Якщо це тривіальна дія, під час якої від користувача не потрібні додаткові дані, то вона виконується і дані синхронізуються з сервером. Якщо ж це менш тривіальна дія, як-от створення нового платежу, то користувачу +надсилається запит на введення даних свого платіжного засобу, а також цілі платежу. Після цього відповідна дія виконується;
- відображення результату. В залежності від наявності проблем під час виконання користувацького запиту, йому може бути відображене або повідомлення про помилку, або результат його дій.

Таблиця 1.1

Характеристика бізнес-процесу «Управління регулярними платежами»

Назва характеристики	Значення характеристики
Ім’я бізнес-процесу	Управління регулярними платежами
Основні учасники	Клієнт
Вхідна подія	Намір клієнта акумулювати в одному застосунку всі дані про різні регулярні платежі, створювати нові регулярні платежі
Вхідний документ	Персональні дані клієнта, банківські дані клієнта
Вихідна подія	Створення масиву користувацьких даних про існуючі та новостворені регулярні платежі
Вихідні документи	Квитанція з оплати створеного регулярного платежу, реєстр користувачів
Клієнт бізнес-процесу	Фізична особа

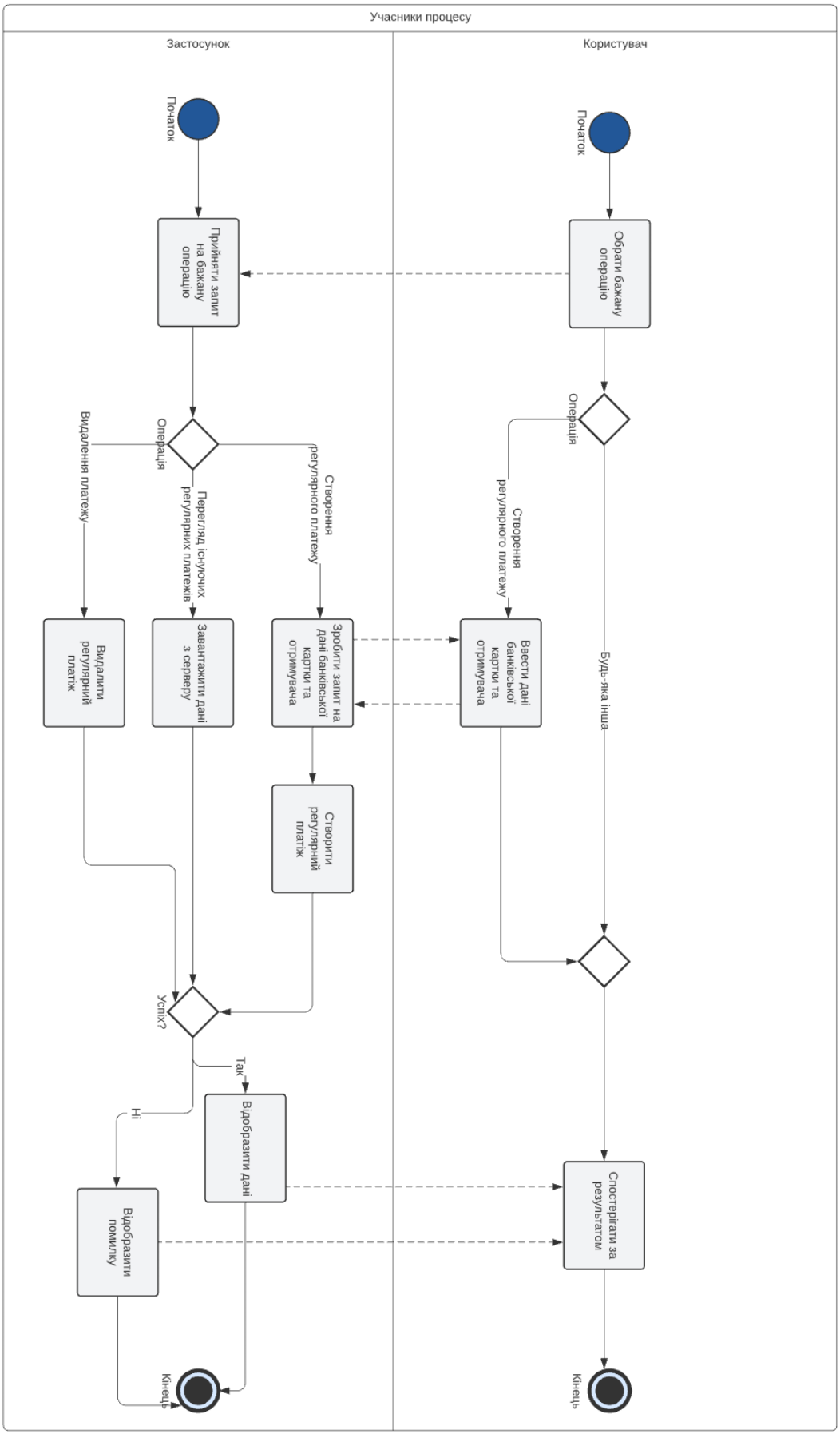

Рис. 1.3. Схема процесу управління регулярними платежами

1.3. Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

На сьогоднішній день програмні засоби, що користуються технологіями, пов’язаними з такими чутливими операціями, як збір інформації про історію транзакцій користувачів, проведення грошових переказів тощо, стикаються з необхідністю швидко адаптуватись до змін законодавства країн або геополітичних союзів, на території яких вони використовуються. Багато підприємств з розробки таких застосунків не готові швидко впроваджувати зміни або розширювати функціональність після оновлення правил користуванням даних з їх предметної області. Прикладом таких змін може бути прийнята у 2018 році Європарламентом оновлена директива PSD2 [7] (оригінальна називається PSD2, Directive (EU) 2015/2366), яка, з-поміж іншого, дозволила (у тому числі розробникам застосунків) збирати дані про транзакції, що були виконані платіжними засобами користувача [8-10]. Але досі можна бачити, що абсолютна більшість застосунків, що спеціалізується на управлінні регулярними платежами, не надає можливості переглядати історію регулярних платежів, асоційованих з платіжним засобом користувача, окрім тих платежів, що користувач вручну надавав застосунку через графічний інтерфейс. Саме через таку відсутність постійного покращення застосунків ринок предметної області залишається відкритим для нових продуктів, що зможуть принести інноваційну функціональність.

У межах огляду і аналізу аналогів буде представлено два застосунки, що є одними з найбільш популярних застосунків з управління регулярними платежами для ОС Android – це «Subby – The Subscription Manager» та «Subscriptions – Manage your regular expenses»[11].

Застосунок «Subscriptions» спеціалізується, судячи за описом та назвою, саме на управління підписками-регулярними платежами. Модуль управління підписками реалізує такі функціональні можливості, як: додавання існуючих, редагування та видалення (з бази даних застосунку) користувачем регулярних платежів вручну; перегляд доданих платежів; нагадування про близкість терміну сплати. Інтерфейс користувача дозволяє зручно і просто оперувати застосунком навіть без використання інструкції користувача. Головним екраном є список регулярних платежів.

Застосунок «Subby» також, судячи за описом і назвою, спеціалізується на управління регулярними платежами. Модуль управління підписками реалізує такі можливості, як: додавання існуючих, редагування регулярних платежів; перегляд доданих платежів; підбиття статистики за платежами; нагадування про близький

термін сплати. Інтерфейс є менш зручним та зрозумілим, ніж у «Subscriptions», але все одно дозволяє без проблем оперувати модулем. Головним екраном також є список регулярних платежів.

Більш детальні відомості за результатами порівняння можна переглянути у табл. 1.2.

Таблиця 1.2.

Порівняльна характеристика програмних продуктів

Фірма-розробник	Simolation	DevNotFoundException
Назва програмного продукту	Subscriptions – Manage your regular expenses	Subby – The Subscription Manager
Версії продукту	Реліз 1.5.2	Реліз 1.8.5
Функціональність	Управління регулярними платежами здійснюється користувачем вручну, а саме: користувач може додавати існуючі (створені у сторонніх сервісах) регулярні платежі, редагувати додані записи, а також видаляти або тимчасово їх архівувати. Застосунок нагадує користувачу про списання коштів, що скоро відбудеться	Управління регулярними платежами здійснюється користувачем вручну, а саме: користувач може додавати існуючі (створені у сторонніх сервісах) регулярні платежі, а також редагувати додані записи (функція видалення тимчасово не працює у всіх користувачів). Застосунок нагадує користувачу про списання коштів, що скоро відбудеться. Відображення статистики за доданими вручну витратами на регулярні платежі відбувається за запитом користувача

Закінчення табл. 1.2

Фірма-розробник	- Simolation	- DevNotFoundException
Інтерфейс користувача	- Перегляд доданих вручну регулярних платежів - додання та редагування доданих платежів - встановлення нагадувань про час сплати - перегляд деталей про платежі	- Перегляд доданих вручну регулярних платежів - додання та редагування доданих платежів - встановлення нагадувань про час сплати - перегляд статистики про додані платежі
Допомога користувачу	Довідка про застосунок з описом політики приватності та контактами розробника	Довідка про ліцензійні дані та назва компанії-розробника
Обсяг банківських даних користувача, що збираються	Тільки дані про платежі, введені користувачем в процесі користування	Тільки дані про платежі, введені користувачем в процесі користування
Наявність локалізації на українську мову	Відсутня	Відсутня
Наявність реклами	Немає	Є
Конвертація валют	Відсутня, платежі відображаються у валюті, що обирається під час першого запуску застосунку	Наявна, під час створення платежів можна вказати у якій вони валюті
Резервне збереження даних	Є локальне та за допомогою google drive	Є, за допомогою google drive

Програмний продукт «Subscriptions» слугує для управління та моніторингу регулярних платежів. Основними варіантами використання цього застосунку є:

- додавання та редагування даних про існуючі регулярні платежі (див. рис. 1.4);

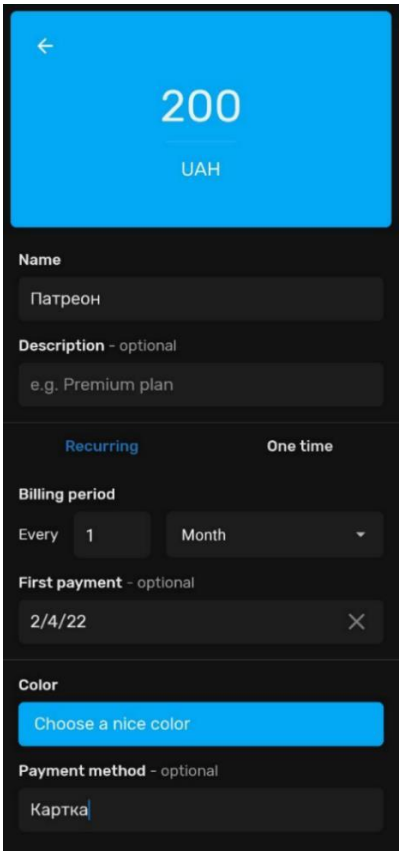

Рис. 1.4. Екранна форма застосунку, призначена для додання даних про існуючу підписку

- перегляд списку доданих регулярних платежів (див. рис. 1.5);

Рис. 1.5. Екранна форма застосунку, призначена для перегляду списку доданих користувачем платежів

- перегляд детальної інформації про платіж (див. рис. 1.6);

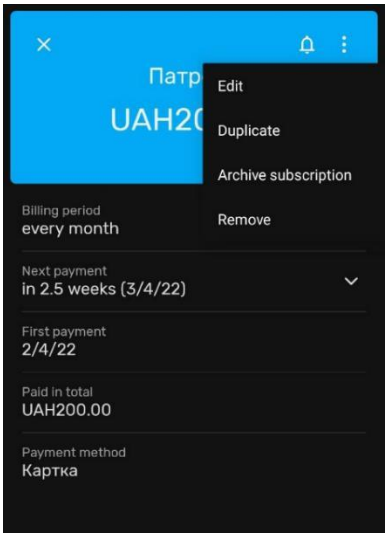

Рис. 1.6. Екранна форма застосунку, призначена для перегляду детальної інформації про платіж, з елементами керування ним

- встановлення нагадування про термін сплати (див. рис. 1.7).

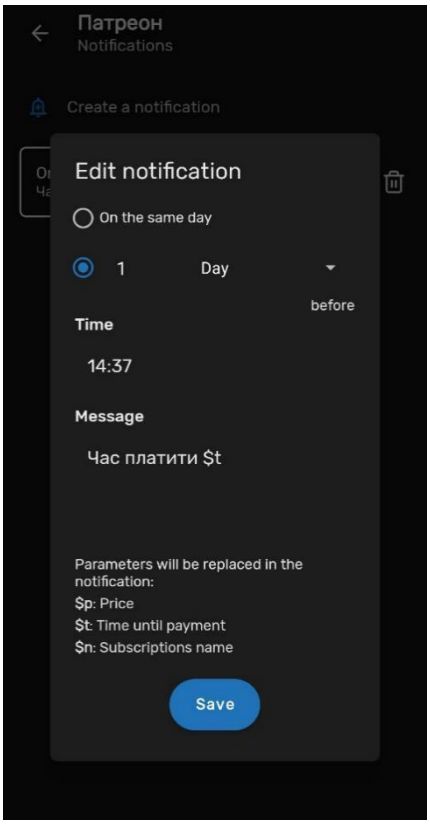

Рис. 1.7. Екранна форма застосунку, призначена для встановлення нагадування про майбутній платіж

Програмний продукт «Subby» слугує для управління регулярними платежами, моніторингу та перегляду статистичної інформації про них. Основними варіантами використання цього застосунку є:

- додавання та редагування даних про існуючі регулярні платежі, а також встановлення нагадування про їх термін оплати (див. рис. 1.8);
- перегляд списку доданих регулярних платежів та інформації про них (див. рис. 1.9);
- перегляд статистичних даних таких як середньомісячні витрати за регулярними платежами (див. рис. 1.10).

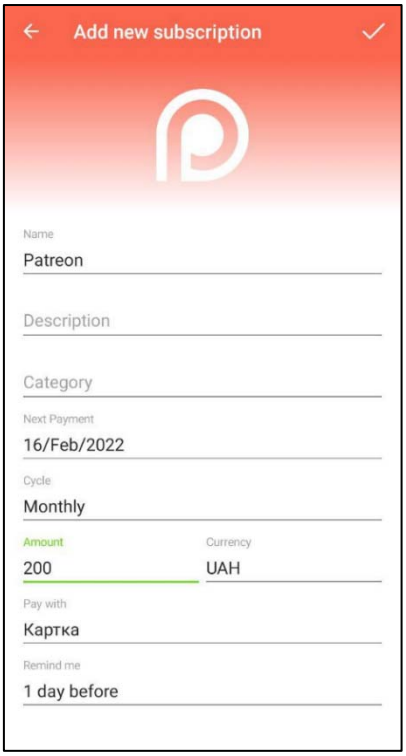

Рис. 1.8. Екранна форма застосунку, призначена для додання даних про існуючий платіж та встановлення нагадування

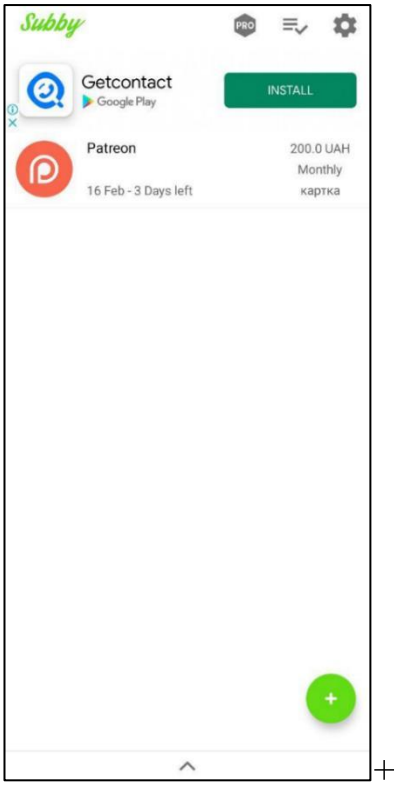

Рис. 1.9. Екранна форма застосунку, призначена для перегляду списку платежів

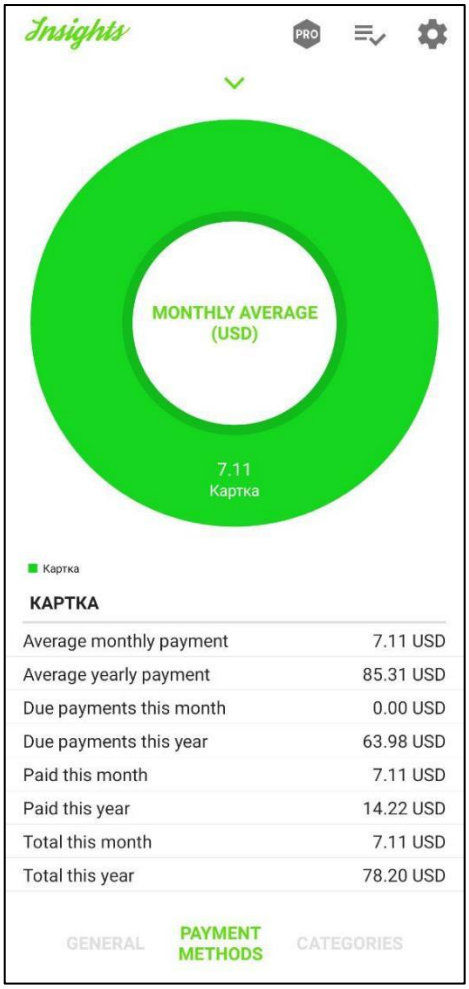

Рис. 1.10. Екранна форма застосунку, призначена для перегляду статистичних даних про платежі (середньомісячні витрати)

Отже, виходячи з проведеного аналізу, обидва розглянутих застосунки-конкуренти, хоч і позиціонують себе як застосунки з управління регулярними платежами, здебільшого є просто інформаційно-статистичними довідниками за даними, що вводяться користувачем. Через це користувачі можуть очікувати від застосунку-конкуренту підтримки такого способу функціонування, коли користувач не бажає або не може надати застосунку дані про свої банківські картки або інші способи створення грошових транзакцій, а додаткова функціональність у вигляді хоча б створення нових регулярних платежів з додатку буде позитивно вирізняти такий застосунок з-поміж наявних аналогів.

2. СПЕЦИФІКАЦІЯ ВИМОГ ДО МОДУЛЯ

2.1. Глосарій

Використовувані протягом цього документу терміни, що належать до досліджуваної предметної області та можуть не бути загальновідомими, зібрані у глосарії. Глосарій поданий у табл. 2.1.

Таблиця 2.1

Глосарій проєкту

Термін	Опис терміну
Основні поняття та категорії предметної області та проєкту	
Застосунок [12]	Користувацька комп'ютерна програма, що дає змогу вирішувати конкретні прикладні задачі користувача [13].
Регулярний платіж	Транзакція на визначену суму, що відбувається з деякою періодичністю.
Транзакція	Будь-яка операція з використанням банківського рахунку, криптогаманця або напряму криптовалютного блокчейну.
Криптотранзакція	Транзакція з використанням криптовалют.
Криптовалюта	Різновид цифрової валюти, емісія та облік якої виконується децентралізованою платіжною системою повністю в автоматичному режимі (без можливості внутрішнього або зовнішнього адміністрування) [14].
Криптогаманець	Сервіс для взаємодії з блокчейн-мережою криптотранзакцій [15].
Блокчейн	Розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), що постійно довшає [16].
Сторонній регулярний платіж	Регулярний платіж, що був створений за межами розроблюваного застосунку.
Тост	Невелике інформаційне повідомлення на дисплеї в ОС Android, зазвичай розташовується внизу екрана [17].
Користувачі системи	
Фізична особа	Особа, яка є учасником правових відносин [18].
Вхідні та вихідні документи	

Закінчення табл. 2.1

Термін	Опис терміну
Деталі транзакції	Дані, необхідні для проведення транзакції, а також її статус (прийнята, в обробці, успішно завершена тощо), подані у вигляді єдиного формалізованого запису
Реєстр користувачів	Запис у базі даних користувачів, що містить персональні дані користувача, а також дані про додані або створені ним регулярні платежі.
Дані криптогаманця	Дані, що дозволяють проводити транзакції з використанням криптогаманця (логін, пароль в середовищі криптогаманця, код двофакторної автентифікації тощо).
Персональні дані користувача	Сукупність відомостей про фізичну особу, яких достатньо для ідентифікації однієї конкретної особи в середовищі застосунку (логін, електронна пошта, тощо)

2.2. Розроблення варіантів використання

2.2.1. Діаграма варіантів використання

Діаграма варіантів використання слугує для концептуального представлення проєктованої системи у вигляді акторів, що взаємодіють з системою, і прецедентів [19], що описують таку взаємодію.

Представлена діаграма має такі варіанти використання:

Реєстрація: користувач реєструється в системі для отримання доступу до функціональності системи.

Авторизація: користувач авторизується в системі для отримання доступу до персональних даних та персоналізованих функцій.

Розрахунок витрат за період: користувач обирає періодичність та валюту, за якими він хоче підрахувати свої періодичні витрати; система підраховує суму грошей за цими критеріями і відображає користувачу.

Перегляд списку платежів: користувач переглядає список створених та сторонніх платежів.

Створення нових платежів: користувач надає через форми застосунку та сервісу, що використовується для проведення платежу, дані, необхідні для створення транзакції, яка буде повторюватись з періодичністю, вказаною користувачем.

Здійснення повторної спроби проведення платежу: користувач здійснює спробу повторити транзакцію, що з деяких причин не була успішною, за потреби надаючи додаткові дані.

Відміна створених платежів: користувач видаляє створений платіж зі списку платежів, тим самим відміняючи його наступне проведення.

Додання інформації про платіж: користувач надає інформацію про сторонній регулярний платіж з метою відстеження цієї інформації у застосунку.

Редагування інформації про платіж: користувач змінює інформацію про сторонній регулярний платіж з метою уточнення відстежуваних у застосунку даних.

Видалення інформації про платіж: користувач видаляє раніше додану інформацію про сторонній регулярний платіж з метою припинення відстеження даного платежу.

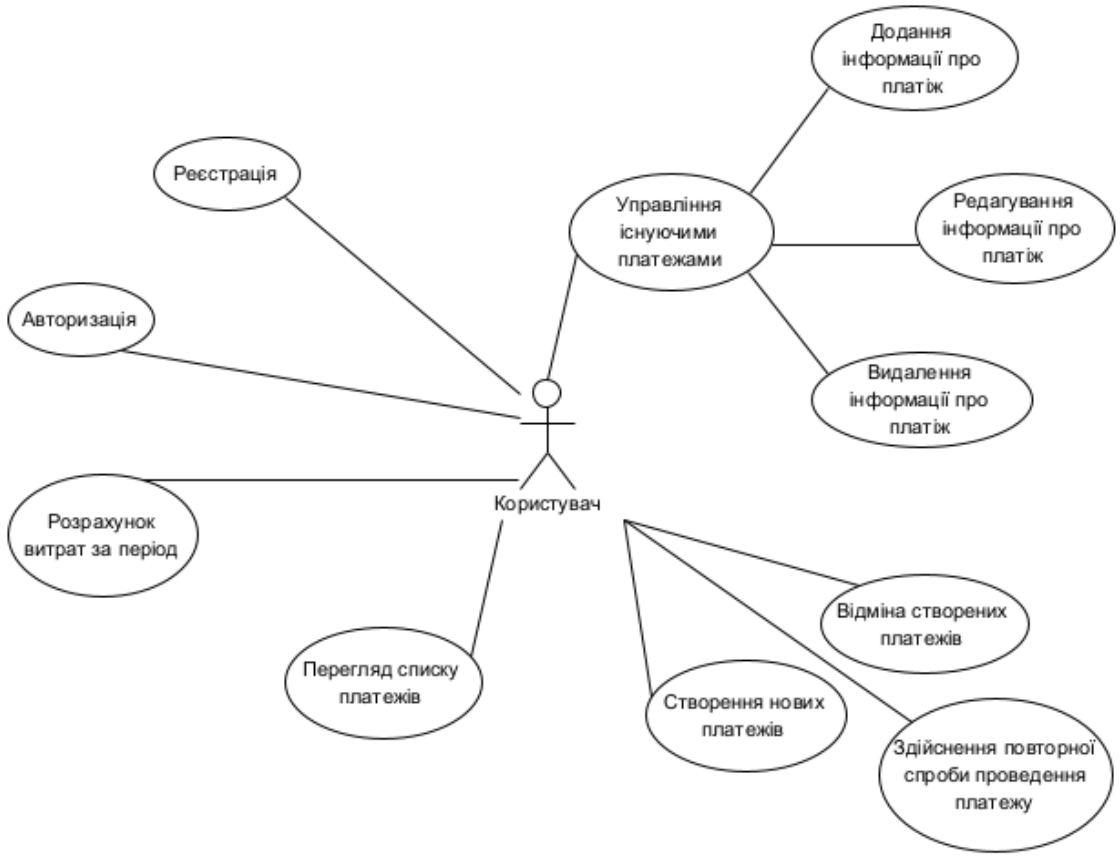

Рис. 2.1. Діаграма варіантів використання

2.2.2. Специфікація варіантів використання

Специфікація варіантів використання націлена на детальний опис всіх наведених раніше прецедентів. В табл. 2.2 – 2.10 наведені такі варіанти використання: зареєструватись, авторизуватись, переглянути список платежів, розрахувати витрати за період, створити регулярний платіж, відмінити створений регулярний платіж, додати інформацію про сторонній регулярний платіж, змінити інформацію про сторонній регулярний платіж, видалити інформацію про сторонній регулярний платіж, здійснити повторну спробу платежу.

Таблиця 2.2

Варіант використання «Зареєструватись»

Характеристика	Значення
Контекст використання	UC-01 Зареєструватись
Дійові особи	Користувач
Передумова	1) Застосунок запущений
Тригер	Користувач має намір користуватися системою
Сценарій	1) Ввести реєстраційні дані в поля на головному екрані застосунку 2) Натиснути кнопку «Зареєструватись»
Постумова	Якщо варіант використання виконався успішно, то користувачу відкриється екран авторизації

Таблиця 2.3

Варіант використання «Авторизуватись»

Характеристика	Значення
Контекст використання	UC-02 Авторизуватись
Дійові особи	Користувач
Передумова	1) Особа користувача зареєстрована
Тригер	Користувач має намір увійти в систему
Сценарій	1) Натиснути «Ввійти» на екрані реєстрації застосунку 2) Ввести логін та пароль 3) Натиснути кнопку «Ввійти»
Постумова	Якщо варіант використання виконався успішно, то користувачу буде представлений список його регулярних платежів

Таблиця 2.4

Варіант використання «Переглянути список платежів»

Характеристика	Значення
Контекст використання	UC-03 Переглянути список платежів
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована
Тригер	Користувач має намір переглянути список регулярних платежів
Сценарій	1) Перейти на головний екран зі списком регулярних платежів
Постумова	Якщо варіант використання виконався успішно, то користувачу буде представлений актуальний список платежів, доданих цим користувачем

Таблиця 2.5

Варіант використання «Розрахувати витрати за період»

Характеристика	Значення
Контекст використання	UC-04 Розрахувати витрати за період
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована
Тригер	Користувач має намір дізнатись сумарні витрати певної періодичності у певній валюті
Сценарій	1) Відкрити меню доступних дій 2) Обрати дію «Розрахувати витрати» 3) Обрати валюту та періодичність 4) Натиснути кнопку «Розрахувати»
Постумова	Якщо варіант використання виконався успішно, то користувачу буде представлено попап-вікно з сумою витрат з вказаною періодичністю

Таблиця 2.6

Варіант використання «Створити регулярний платіж»

Характеристика	Значення
Контекст використання	UC-05 Створити регулярний платіж
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована 2) Особа користувача авторизована на сервісі, за допомогою якого будуть здійснюватись платежі
Тригер	Користувач має намір здійснювати платежі на регулярній основі

Закінчення табл. 2.6

Характеристика	Значення
Сценарій	1) Відкрити меню доступних дій 2) Обрати дію «Створити» 3) Ввести потрібні параметри платежу та обрати сервіс 4) Натиснути кнопку «Створити» 5) Провести автентифікацію на сервісі, на який користувача буде переадресовано
Постумова	Якщо варіант використання виконався успішно, то користувачу буде показано список регулярних платежів, у тому числі з новостворeним; платіж буде проводитись з вказаною регулярністю

Таблиця 2.7

Варіант використання «Відмінити створений регулярний платіж»

Характеристика	Значення
Контекст використання	UC-06 Відмінити створений регулярний платіж
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована 2) За допомогою застосунку було створено дійсний досі регулярний платіж
Тригер	Користувач має намір відмінити наступні виплати за регулярним платежем
Сценарій	1) Перейти на головний екран зі списком платежів 2) На елементі, що відповідає бажаному платежу, обрати піктограму відміни
Постумова	Якщо варіант використання виконався успішно, то регулярний платіж буде відмінено і видалено зі списку регулярних платежів на головному екрані

Таблиця 2.8

Варіант використання «Додати інформацію про сторонній регулярний платіж»

Характеристика	Значення
Контекст використання	UC-07 Додати інформацію про сторонній регулярний платіж
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована
Тригер	Користувач має намір внести в застосунок дані про регулярний платіж, що був створений за допомогою стороннього сервісу

Закінчення табл. 2.8

Характеристика	Значення
Сценарій	1) Відкрити меню доступних дій 2) Обрати дію «Імпортувати» 3) Ввести потрібні дані платежу
Постумова	Якщо варіант використання виконався успішно, то в списку регулярних платежів на головному екрані застосунку буде додано дані про імпортований платіж

Таблиця 2.9

Варіант використання «Змінити інформацію про сторонній регулярний платіж»

Характеристика	Значення
Контекст використання	UC-08 Змінити інформацію про сторонній регулярний платіж
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована 2) У застосунок було додано інформацію про сторонній регулярний платіж
Тригер	Користувач має намір відредагувати раніше внесені дані про сторонній регулярний платіж
Сценарій	1) Перейти на головний екран зі списком платежів 2) На елементі, що відповідає бажаному платежу, обрати піктограму редагування 3) Ввести потрібні дані у відповідні поля у вікні, що з’явилось 4) Натиснути кнопку «Зберегти»
Постумова	Якщо варіант використання виконався успішно, то у списку регулярних платежів на головному екрані буде відображено оновлені дані про відредагований платіж

Таблиця 2.10

Варіант використання «Видалити інформацію про сторонній регулярний платіж»

Характеристика	Значення
Контекст використання	UC-09 Видалити інформацію про сторонній регулярний платіж
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована 2) У застосунок було додано інформацію про сторонній регулярний платіж
Тригер	Користувач має намір видалити раніше внесені дані про сторонній регулярний платіж

Закінчення табл. 2.10

Характеристика	Значення
Сценарій	1) Перейти на головний екран зі списком платежів 2) На елементі, що відповідає бажаному платежу, обрати піктограму видалення
Постумова	Якщо варіант використання виконався успішно, то обраний регулярний платіж буде видалено зі списку регулярних платежів на головному екрані

Таблиця 2.11

Варіант використання «Здійснити повторну спробу платежу»

Характеристика	Значення
Контекст використання	UC-10 Здійснити повторну спробу платежу
Дійові особи	Користувач
Передумова	1) Особа користувача авторизована 2) Під час створення регулярного платежу виникла помилка
Тригер	Користувач має намір спробувати провести оплату ще раз
Сценарій	1) Перейти на головний екран зі списком платежів 2) На елементі, що відповідає бажаному платежу, обрати піктограму перегляду 3) У вікні, що відкрилось, натиснути кнопку «Спробувати ще раз», за потреби ввести додаткові дані
Постумова	Якщо варіант використання виконався успішно, то користувачу буде показано результат нової спроби оплати обраного регулярного платежу

2.3. Специфікація функціональних та нефункціональних вимог

2.3.1. Функціональні вимоги

Функціональні вимоги — це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку: обчислення даних, маніпулювання даними, обробка даних та інші специфічні функції, які має виконувати система. У даному випадку контактом завжди є розробник, тому ця колонка буде вилучена. Всі функціональні вимоги та їх атрибути вимог наведено у табл. 2.12.

Таблиця 2.12

Специфікація функціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог	
		Пріоритет	Важкість
FR-UC-01	Зареєструвати користувача	Обов’язкова	Висока
FR-UC-02	Авторизувати користувача	Обов’язкова	Середня
FR-UC-03	Переглянути список регулярних платежів	Обов’язкова	Низька
FR-UC-04	Розрахувати витрати за період	Рекомендована	Низька
FR-UC-05	Створити регулярний платіж	Рекомендована	Висока
FR-UC-06	Відмінити створений регулярний платіж	Рекомендована	Висока
FR-UC-07	Додати інформацію про існуючий регулярний платіж	Обов’язкова	Низька
FR-UC-08	Змінити інформацію про існуючий регулярний платіж	Обов’язкова	Низька
FR-UC-09	Видалити інформацію про існуючий регулярний платіж	Обов’язкова	Низька
FR-UC-10	Здійснити повторну спробу платежу	Рекомендована	Середня

2.3.2. Нефункціональні вимоги

Нефункціональні вимоги — це вимоги до програмного забезпечення, які задають критерії для оцінки якості його роботи. В табл. 2.13 вказані нефункціональні вимоги, які є суттєвими для розроблюваного модулю.

Таблиця 2.13

Специфікація нефункціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Важкість	Контакт
Застосовність				
AR-01	Період ознайомлення з застосунком для нових користувачів – 10 хвилин	Обов’язкова	Середня	Розробник
AR-02	Час застосування змін – до 5 секунд	Рекомендована	Низька	Розробник

Продовження табл. 2.13

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Важкість	Контакт
AR-03	Зарахування коштів на рахунок отримувача – до 5 робочих днів	Обов’язкова	Низька	Розробник, служба підтримки обраного сервісу для здійснення платежів
Надійність				
RR-01	Для роботи з системою повинна здійснюватися реєстрація та подальша авторизація користувача	Обов’язкова	Висока	Розробник
RR-02	Стійкість до збоїв та можливість продовжити роботу з системою у випадку збою	Обов’язкова	Середня	Розробник
RR-03	Доступність – від 95% доступного часу	Рекомендована	Середня	Розробник
Робочі характеристики				
PR-01	Час розрахунку витрат за певний період – не більше 2 секунд	Обов’язкова	Низька	Розробник
PR-02	Час обробки запиту на редагування даних – не більше 5 секунд	Рекомендована	Середня	Розробник
PR-03	Можливість одночасного користування системою 20 користувачами без зниження продуктивності	Рекомендована	Низька	Розробник
PR-04	Час обробки запиту на створення регулярного платежу – не більше 5 секунд	Рекомендована	Низька	Розробник

Продовження табл. 2.13

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Важкість	Контакт
Експлуатаційна придатність				
OR-01	Для експлуатації розроблюваної системи має бути забезпечено безперебійне з'єднання з мережею Інтернет	Обов'язкова	Висока	Інтернет-провайдер, Розробник
OR-02	Сервер розміщення Web-сайту повинен відповідати мінімальним потребам бази даних; мати можливість прийому та відправки даних протоколом HTTP	Обов'язкова	Середня	Розробник
Проєктні обмеження				
PR-01	Мови програмування: Java, Kotlin, Python	Рекомендована	Низька	Розробник
PR-02	Інструментальні засоби розробки: Android Studio, PyCharm	Рекомендована	Низька	Розробник
PR-03	Технології розробки: Django, Android SDK, Room	Рекомендована	Середня	Розробник
PR-04	Технології роботи з БД: Django SQLite	Рекомендована	Низька	Розробник
PR-05	Додаткові засоби розробки: CSS, HTML, XML, JSON	Рекомендована	Низька	Розробник
Інтерфейси				
Апаратні інтерфейси				
IH-01	Протокол обміну даними між клієнтом та сервером [1] – HTTP	Обов'язкова	Середня	Розробник
Програмні інтерфейси				
IS-01	Система взаємодіє з СКБД SQLite	Обов'язкова	Низька	Розробник

Закінчення табл.2.13

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Важкість	Контакт
Комунікаційні інтерфейси				
CI-01	Система взаємодіє з «Coinbase API»	Обов’язкова	Висока	Розробник
Інтерфейси користувача				
IU-01	Користувачу надаються повідомлення про помилку у вигляді повідомлень типу «тост»	Обов’язкова	Низька	Розробник
IU-02	Система надає користувачу вказівки у вигляді валідаційних повідомлень під кожним полем, що має обмеження	Обов’язкова	Низька	Розробник
Застереження щодо питань, пов’язаних з авторськими правами				
CR-01	Авторські права на розроблену систему належать розробнику	Обов’язкова	Середня	Розробник

2.4. Проєктування інтерфейсу користувача

Вайрфрейми основних екранів графічного інтерфейсу користувача, що відповідають варіантам використання, зображено на рис 2.2 - 2.11.

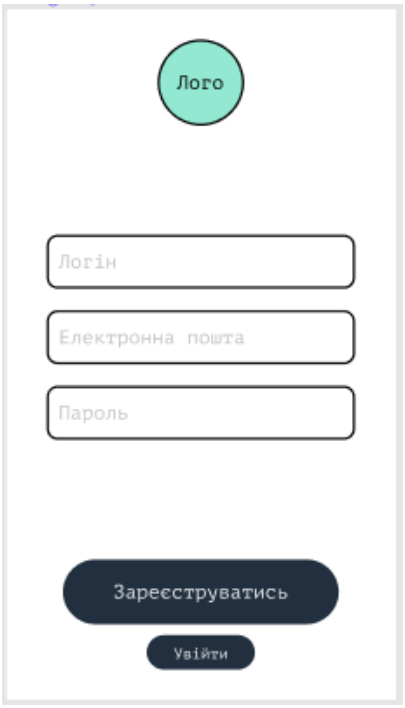

Рис. 2.2. Мокап варіанту використання «Зареєструвати користувача»

Рис. 2.3. Мокап варіанту використання «Авторизувати користувача»

Рис. 2.4. Мокап варіанту використання «Переглянути список регулярних платежів»

Рис. 2.5. Мокап варіанту використання «Розрахувати витрати за період»

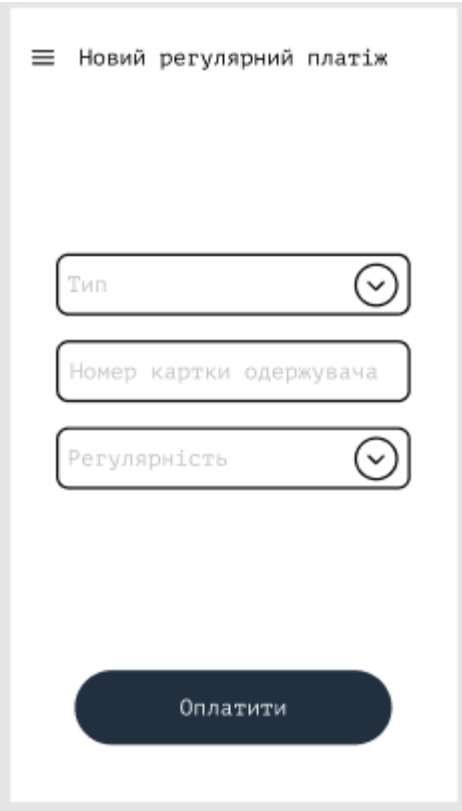

Рис. 2.6. Мокап варіанту використання «Створити регулярний платіж»

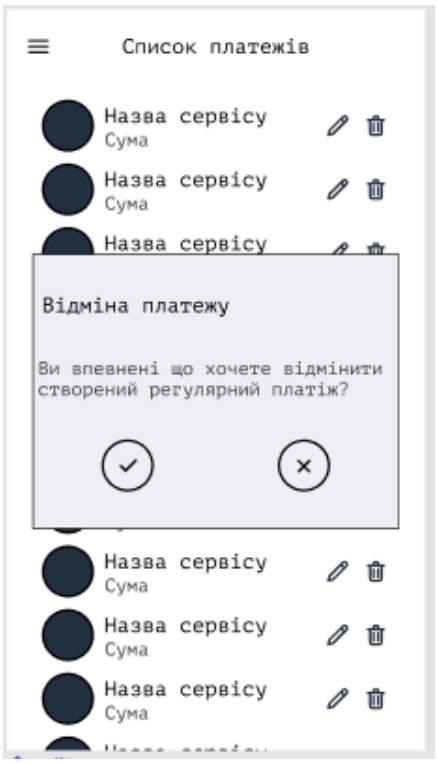

Рис. 2.7. Мокап варіантів використання «Відмінити створений регулярний платіж»

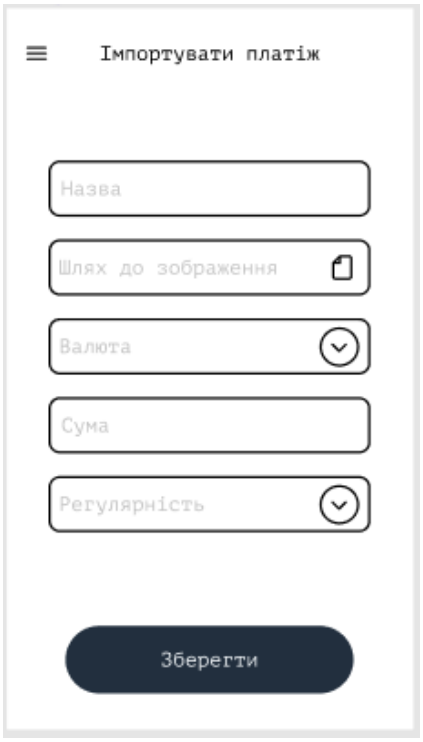

Рис. 2.8. Мокап варіанта використання «Додати інформацію про сторонній регулярний платіж»

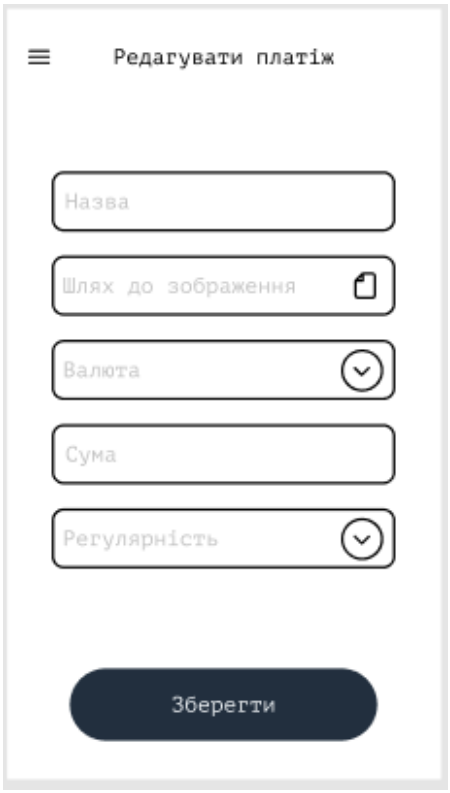

Рис. 2.9. Мокап варіанта використання «Змінити інформацію про сторонній регулярний платіж

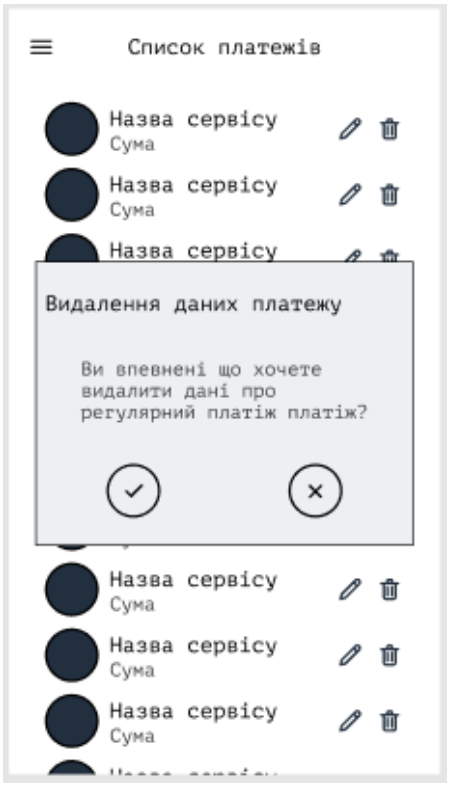

Рис. 2.10. Мокап варіанта використання «Видалити інформацію про сторонній регулярний платіж»

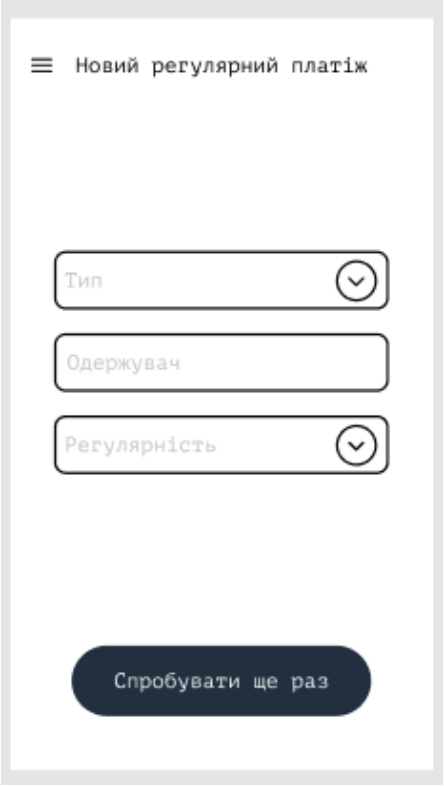

Рис. 2.11. Мокап варіанта використання «Здійснити повторну спробу платежу»

За результатами другого розділу було визначено функціональні та нефункціональні вимоги до системи, а також спроєктовано інтерфейс користувача.

3. ПРОЄКТНІ ТА ТЕХНІЧНІ РІШЕННЯ

3.1. Логічна постановка

Список формалізованих задач, розв’язати які має застосунок «RegPay», складається з наступних пунктів:

1. перегляд сторонніх та створених у застосунку регулярних платежів;
2. додання даних стороннього платежу;
3. редагування даних стороннього платежу;
4. видалення даних стороннього платежу;
5. створення нового платежу;
6. відміна створеного платежу;
7. повторна спроба проведення регулярного платежу у разі неуспіху;
8. розрахунок витрат у певній валюті за певний період.

Математичне формулювання має тільки останній пункт. Його можна описати наступною формулою:

$$C_в = \sum B_п ,$$

де $C_в$ – це сума витрат за певний період;

$B_п$ – це витрати у вказаній валюті за платежем, обчислені за певний період.

Для інших пунктів далі наведено опис логіки послідовних дій.

Після того, як користувач проходить авторизацію, йому відображається головна сторінка з повним списком основних даних (назва, картинка, валюта та сума) регулярних платежів, закріплених за цим користувачем. На елементах списку є одна або дві кнопки керування даним елементом. Якщо платіж є стороннім, на ньому відображаються клавіші редагування даних про платіж (що запускає спеціальний екран для введення змінених даних та збереження змін) та видалення цих даних (що запускає попап-вікно з проханням до користувача підтвердити свій намір). Якщо платіж був створений у застосунку, на елементі відображається кнопка скасування цього платежу (що також викликає попап-вікно з проханням до користувача підтвердити свої наміри), а також, якщо платіж не був успішним, кнопка повторної спроби платежу (що відкриває спеціальне вікно з продемонстрованими даними, які користувач вводив у процесі створення платежу і кнопкою повторної спроби, а також, за необхідності, з додатковими полями для додаткових даних користувача). На кожному екрані після авторизації у застосунку

доступне меню дій, у якому можна обрати такі дії, як «імпорт», що відкриває екран з формою додання даних про сторонній платіж, «створення», що відкриває екран з формою для введення даних для створення нового регулярного платежу та «розрахунок витрат», що відкриває екран з формою для визначення параметрів розрахунку (періодичності та валюти), і, після вибору, відображає суму у попап-вікні. Дані про всі операції миттєво синхронізуються з сервером, а список і параметри платежів у додатку також миттєво оновлюються. Це дозволяє використовувати застосунок на будь-якому пристрої без потреби перенесення даних вручну.

3.2. Проєктування структури бази даних

3.2.1. Концептуальне інфологічне проєктування

Спільний словник даних для клієнтської та серверної [1] частини застосунку наведений у табл. 3.1.

Таблиця 3.1

Словник даних

№ з/п	Найменування елемента	Ідентифікатор	Тип і довжина	Призначення елемента
1	Ідентифікатор користувача	id	Int	Фактичне
2	Ідентифікатор додаткових даних користувача	id	Int	Фактичне
3	Ідентифікатор налаштування системи	id	Int	Фактичне
4	Ідентифікатор регулярного платежу	id	Int	Фактичне
5	Назва системного налаштування	key	Nvarchar(100)	Фактичне
6	Дані системного налаштування	value	CLOB	Фактичне
7	Логін користувача	username	Nvarchar(150)	Фактичне
8	Хеш паролю користувача	password	Nvarchar(150)	Фактичне

Закінчення табл. 3.1

№ з/п	Найменування елемента	Ідентифікатор	Тип і довжина	Призначення елемента
9	Електронна пошта користувача	email	Nvarchar(150)	Фактичне
10	Назва регулярного платежу	name	Nvarchar(50)	Фактичне
11	URL картинки для регулярного платежу	picUrl	Nvarchar(2000)	Фактичне
12	Валюта регулярного платежу	currency	Int	Фактичне
13	Сума регулярного платежу	amount	Double	Фактичне
14	Періодичність регулярного платежу	regularity	Int	Фактичне
15	Сервіс (криптогаманець), за допомогою якого проводиться платіж	service	Int	Фактичне
16	Статус останньої спроби проведення платежу	status	Int	Фактичне
17	Отримувач платежу	destination	CLOB	Фактичне
18	Дата останньої спроби проведення платежу	dateString	Nvarchar(20)	Фактичне
19	Назва додаткових даних користувача	key	Nvarchar(100)	Фактичне
20	Додаткові дані користувача	data	CLOB	Фактичне

Обмеження атрибутів сутностей наведені у табл. 3.2.

Таблиця 3.2

Обмеження атрибутів

№	Ім’я атрибуту або агрегату	Межі / допустимі значення	Структура (формат)	Умова	Значення за замовчуванням
Клієнтська частина					
1	User.id	Більше 0		UNIQUE, NOT NULL	
2	username			NOT NULL	
3	password			NOT NULL	
4	SystemSetting.id	Більше 0		UNIQUE, NOT NULL	
5	name			NOT NULL	
6	value			NOT NULL	
7	RegularPayment.id	Більше 0		UNIQUE, NOT NULL	
8	name			NOT NULL	
9	picUrl		URL	NOT NULL	
10	currency			NOT NULL	
11	amount			NOT NULL	
12	regularity			NOT NULL	
13	service			NOT NULL	
14	status			NOT NULL	
15	destination				
Серверна частина					
16	User.id	Більше 0		UNIQUE, NOT NULL	
17	username	Від 1 до 150		NOT NULL	
18	password	Від 1 до 150		NOT NULL	
19	first_name	Від 1 до 150			
20	last_name	Від 1 до 150			
21	email	Від 1 до 150			
22	is_staff			NOT NULL	
23	is_active			NOT NULL	
24	is_superuser			NOT NULL	
25	last_login			NOT NULL	
26	date_joined			NOT NULL	
27	RegularPaymentModel.id	Більше 0		UNIQUE, NOT NULL	
28	name	Від 1 до 50		NOT NULL	

Закінчення табл.3.2

№	Ім’я атрибуту або агрегату	Межі / допустимі значення	Структура (формат)	Умова	Значення за замовчуванням
29	picUrl	Від 1 до 2000	URL		
30	currency			NOT NULL	
31	amount			NOT NULL	
32	regularity			NOT NULL	
33	service			NOT NULL	
34	status			NOT NULL	1
35	destination				
36	dateString	Від 1 до 20	дд/мм/рррр		
37	SystemSettingModel.key	Від 1 до 100		UNIQUE, NOT NULL	
38	value			NOT NULL	
39	UserDataModel.id	Більше 0		UNIQUE, NOT NULL	
40	key	Від 1 до 100		UNIQUE	
41	data			NOT NULL	

3.2.2. Проєктування логічної моделі бази даних

У якості інструмента проєктування бази даних було використано CASE-засіб Visual Paradigm, а саме його підмодуль для проєктування ERD, який дозволив забезпечити всі необхідні властивості бази даних, такі як функціональна повнота; мінімальна надмірність; цілісність бази даних (таблична, посилальна цілісність, забезпечувана ключами і тригерами тощо); узгодженість; актуальність; безпека; відновлюваність; логічна та фізична незалежність; ефективність.

У ролі СКБД було обрано SQLite як надлегку та зручну СКБД, підтримка якої інструментами для абстракції керування базою даних широко розповсюджена [20].

Логічна модель бази даних зображена на рис. 3.1.

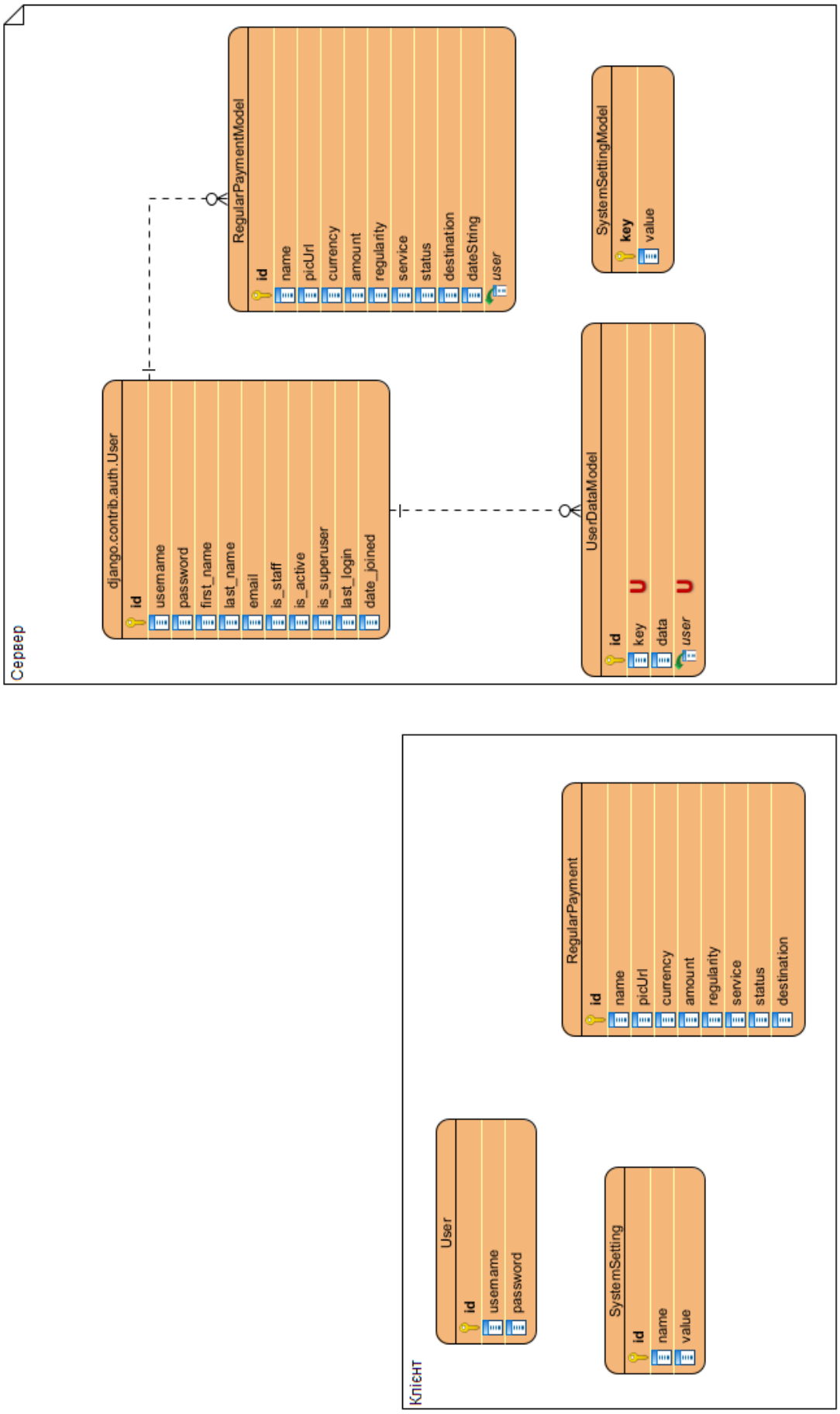

Рис. 3.1. Логічна модель бази даних

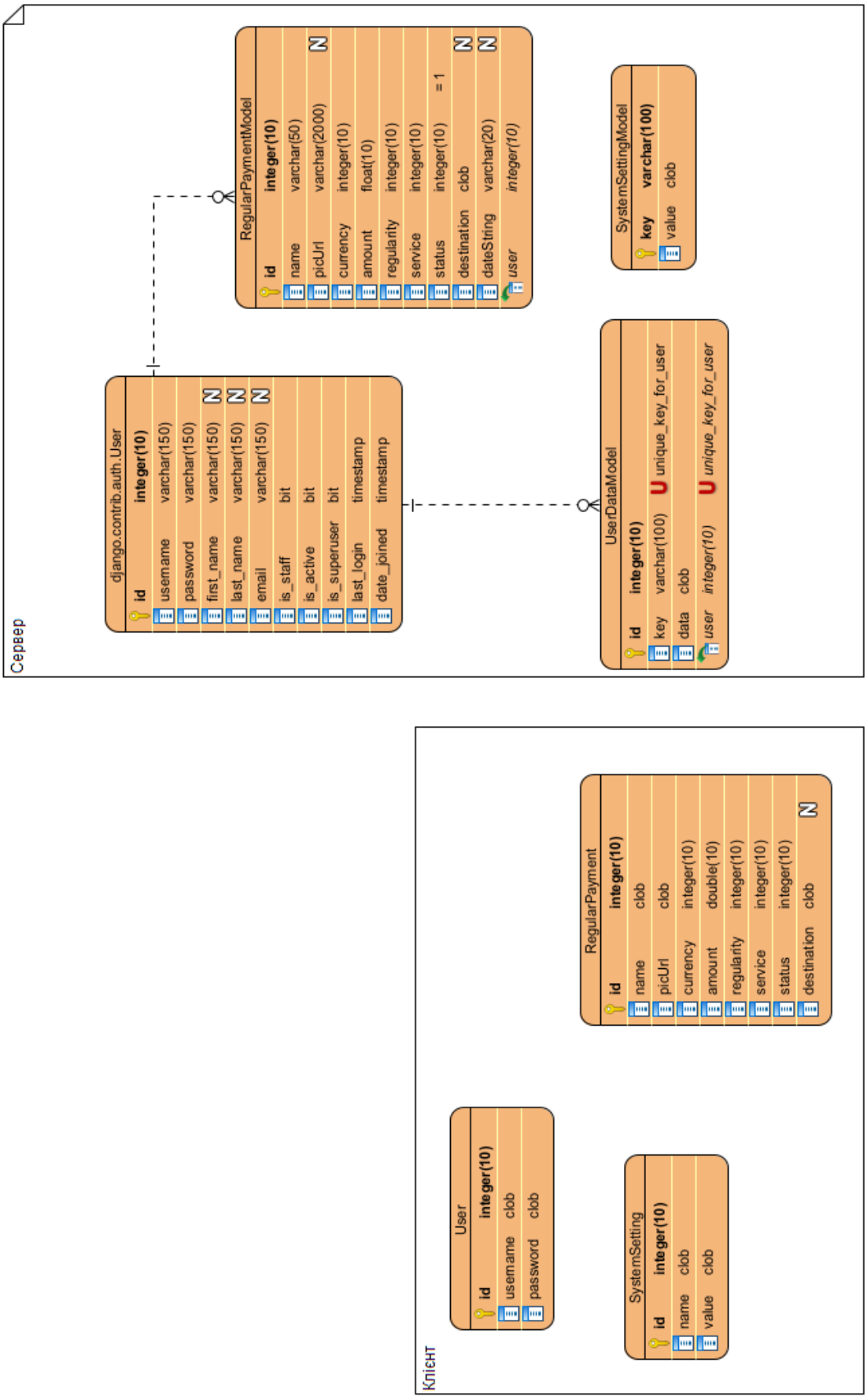

Рис. 3.2. Фізична модель бази даних

3.2.3. Проєктування фізичної моделі бази даних

Для формування фізичної моделі БД також використовувався інструмент Visual Paradigm. Для програмної реалізації було використано засоби абстракції від прямого використання СКБД, різні для клієнту та серверу [1]. Для клієнту використовувалась бібліотека персистентності Room, що є частиною Android Jetpack [21]. Для серверу використовувались засоби фреймворку Django 3.2.

Фізична модель бази даних зображена на рис. 3.2.

3.3. Розроблення архітектури програмної системи

3.3.1. Діаграма класів

Під час розробки серверної частини застосунку будо використано шаблон об’єктно-орієнтованого програмування під назвою MVC (Model-View-Controller) [22], а точніше – його реалізацію в мові програмування Python за допомогою фреймворку Django, названу MVT(Model-View-Template). Класи Моделі (Model) відповідають за зберігання та базову обробку даних, у той же час як класи Вигляду (View) відповідають за операції над цими даними, для відображення користувачу результати їх обробки, стаючи «мостом» між даними та Шаблонами (Template), які займаються графічним відображенням користувачу [23].

Так як сервер не має інтерфейсу користувача, а тільки обробляє дані клієнтського застосунку, єдиним класом шаблонів є клас-заглушка LogInForm, що не виконує ніяких функцій окрім підтримки архітектури MVT.

До класів типу View відносяться класи MainView (котрий займається взаємодією з клієнтським застосунком протоколом HTTP) та CompleteAuthView (котрий слугує для єдиної цілі – показати користувачу успішний результат авторизації на сервісі Coinbase, за допомогою якого у застосунку здійснюються регулярні платежі). Також сюди можна включити клас Logger, що займається форматуванням та записом логів серверної частини застосунку.

До класів типу Model відносяться класи django.contrib.auth.User (не реалізований самостійно, але широко використовуваний у проєкті клас користувача з бібліотеки Django), UserDataModel (для зберігання та базової обробки додаткових даних, прив’язаних до користувача), SystemSettingModel (для зберігання системних налаштувань) та RegularPaymentModel (для зберігання даних про регулярні платежі). Також сюди можна віднести клас UserManager, що спрощує операції над таблицею User в базі даних.

Далі наведено основні класи клієнтської частини застосунку та їх опис.

User – клас, що зберігає дані сутності користувача в БД.

SystemSetting – клас, що зберігає дані сутності системного налаштування в БД.

RegularPayment – клас, що зберігає дані сутності регулярного платежу в БД.

UserDao – клас, що дозволяє оперувати над об’єктами користувача в БД.

SystemSettingDao – клас, що дозволяє оперувати над об’єктами системних налаштувань в БД.

RegularPaymentDao – клас, що дозволяє оперувати над об’єктами регулярних платежів в БД.

AppDatabase – клас, що дозволяє отримати доступ до класів, що оперують над сутностями в БД.

RegularPaymentModel – клас, що містить (та може самостійно завантажувати) список регулярних платежів, завантажених з БД, для використання у класах RegularPaymentHolder та RegularPaymentAdapter.

RegularPaymentHolder – клас, що може оперувати над окремими елементами списку регулярних платежів, що відображається на головному екрані застосунку.

RegularPaymentAdapter – клас, що виконує функції прошарку між даними класу RegularPaymentHolder та списком на головному екрані застосунку, а також може управляти цим списком як одним цілим.

ServerConnectionHandler – клас, що виконує основну взаємодію з сервером за допомогою протоколу HTTP.

SignUpActivity – клас, що відповідає за екран реєстрації в застосунку.

LogInActivity – клас, що відповідає за екран авторизації в застосунку.

NavDrawerTestActivity – клас, що відповідає за відображення меню у верхній частині всіх інших клієнтських екранів, а також здійснює основну взаємодію інших екранів застосунку з навколишньою системою.

ImportFragment – клас, що відповідає за наповнення, відображення та взаємодію з екраном додання стороннього платежу.

ListFragment – клас, що відповідає за наповнення, відображення та взаємодію з екраном, на якому показано список регулярних платежів.

EditFragment – клас, що відповідає за наповнення, відображення та взаємодію з екраном редагування стороннього платежу.

CreateFragment – клас, що відповідає за наповнення, відображення та взаємодію з екраном створення нового платежу, а також повторної спроби його проведення.

CalcSpendingsFragment – клас, що відповідає за наповнення, відображення та взаємодію з екраном підрахунку витрат у валюті за період, а також за цей підрахунок.

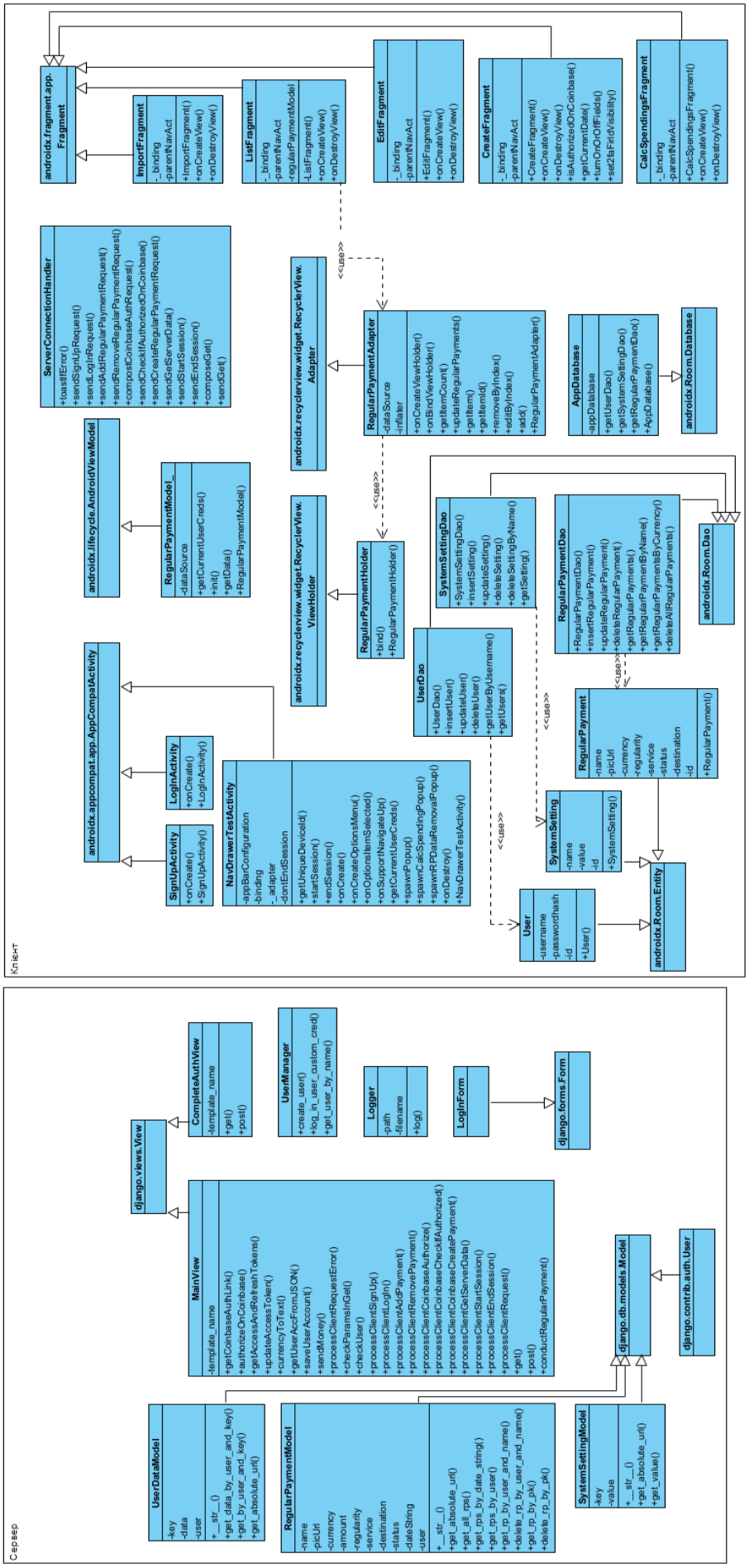

Рис. 3.3. UML-діаграма класів

3.3.2. Діаграми станів

Діаграми станів основних варіантів використання зображено на рис. 3.4 - 3.7.

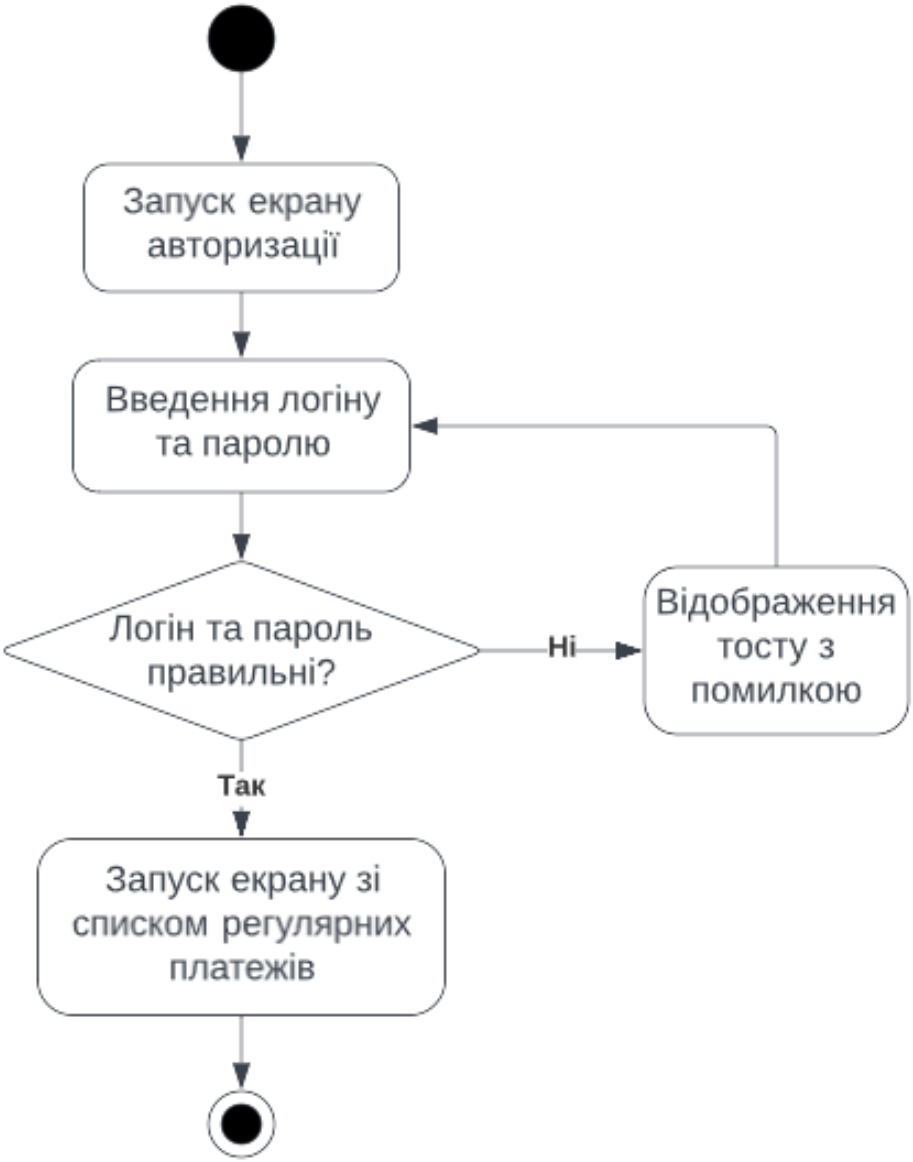

Рис. 3.4. Діаграма станів варіанту використання «Переглянути список регулярних платежів»

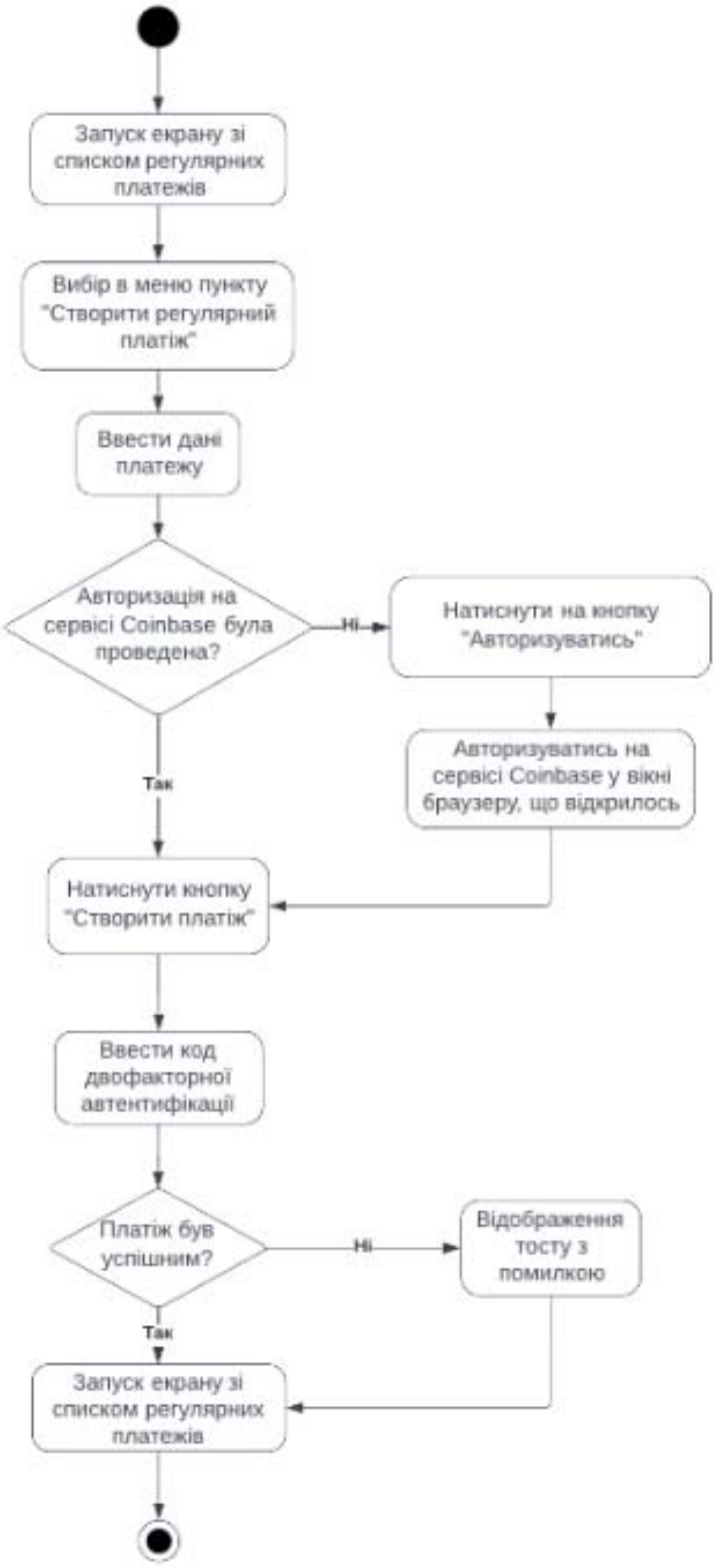

Рис. 3.5. Діаграма станів варіанту використання «Створити регулярний платіж»

Рис. 3.6. Діаграма станів варіанту використання «Додати інформацію про сторонній регулярний платіж»

Рис. 3.7. Діаграма станів варіанту використання «Розрахувати витрати за період»

3.4. Тестування програмної системи

Тестування модулю «Розроблення мобільного застосунку для здійснення регулярних платежів» було проведене вручну; надалі описана процедура проведеного системного тестування.

Тест-вимоги:

1. авторизація користувача;
    a. перевірити авторизацію з неправильно введеними даними;
2. управління сторонніми платежами;
    a. перевірити наявність платежу в списку після додання;
    b. перевірити список платежів після редагування;
    c. перевірити список платежів після видалення;
    d. перевірити створення платежу з назвою існуючого платежу;
3. створення платежів;
    a. перевірити список платежів після створення платежу;
    b. перевірити статус платежу у списку після неуспішного створення;
4. розрахунок витрат;

a. перевірити суму витрат за створеними платежами.

У табл. 3.3-3.6 наведені тестові плани з відразу доданими звітами про проведення тестування.

Тестовий приклад: 1. авторизація користувача.

Тест-вимоги, що перевіряються: 1. а.

Критерій проходження тесту: отримані результати збігаються з очікуваними.

Тест-план зі звітом наведений у табл. 3.3.

Таблиця 3.3

Тест-план «Авторизація користувача»

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	Спроба авторизуватись з неправильно вказаним логіном	Відображено тост про помилку входу з закликом перевірити введені дані. Користувача не авторизовано.	Було відображено тост про помилку входу з закликом перевірити введені дані. Користувача не було авторизовано (рис. 3.8).	Так
2	Спроба авторизуватись з неправильно вказаним паролем	Відображено тост про помилку входу з закликом перевірити введені дані. Користувача не авторизовано.	Було відображено тост про помилку входу з закликом перевірити введені дані. Користувача не було авторизовано.	Так

Відмітка про проходження: пройдений.

Рис. 3.8. Спроба авторизуватись при неправильному введеному імені користувача

Тестовий приклад: 2. управління сторонніми платежами.
Тест-вимоги, що перевіряються: 2. a, 2. b, 2. c, 2. d.
Критерій проходження тесту: отримані результати збігаються з очікуваними.
Тест-план зі звітом наведений у табл. 3.4.

Таблиця 3.4

Тест-план «Управління сторонніми платежами»

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	Додати сторонній регулярний платіж шляхом обрання в меню кнопки «Імпортувати», внесення даних та натискання кнопки «Зберегти»	Користувачу відображається головний екран зі списком регулярних платежів, у якому присутній доданий платіж.	Користувачу відобразився головний екран зі списком регулярних платежів, у якому присутній доданий платіж.	Так

Закінчення табл. 3.4

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
2	Відредагувати суму оплати стороннього регулярного платежу шляхом натискання піктограми редагування на елементі списку з платежем, внесення нової суми оплати та натискання кнопки «Зберегти»	Користувачу відображається головний екран зі списком регулярних платежів, у якому на елементі, що відповідає редагованому платежу, відображається нова сума оплати.	Користувачу відобразився головний екран зі списком регулярних платежів, у якому на елементі, що відповідає редагованому платежу, відображається нова сума оплати.	Так
3	Видалити сторонній регулярний платіж шляхом натискання піктограми виділення на відповідному елементі списку, а також підтвердження такої дії у вспливаючому вікні.	Користувачу відображається головний екран зі списком регулярних платежів, у якому попередньо видалений елемент відсутній.	Користувачу відобразився головний екран зі списком регулярних платежів, у якому попередньо видалений елемент відсутній.	Так
4	Спробувати створити регулярний платіж з назвою, яка використовується іншим регулярним платежем	Користувачу відображається повідомлення про те, що платіж з такою назвою вже існує.	Користувачу відобразилось повідомлення про те, що платіж з такою назвою вже існує (рис. 3.9).	Так

Відмітка про проходження: пройдений.

Рис. 3.9. Спроба створити регулярний платіж з назвою, що вже використовується платежем

Тестовий приклад: 3. створення платежів.
Тест-вимоги, що перевіряються: 3. a, 3. b.
Критерій проходження тесту: отримані результати збігаються з очікуваними.
Тест-план зі звітом наведений у табл. 3.5.

Таблиця 3.5

Тест-план «Створення платежів»

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	Спроба створити регулярний платіж шляхом обрання в меню пункту «Створити», проходження авторизації на сервісі Coinbase, введення потрібних даних і натискання на кнопку «Створити».	Користувачу відображається головний екран зі списком платежів; новий платіж присутній у списку.	Користувачу відобразився головний екран зі списком платежів; новий платіж був присутній у списку.	Так
2	Спроба створити регулярний платіж шляхом обрання в меню пункту «Створити», проходження авторизації на сервісі Coinbase, введення потрібних даних (у тому числі, суми, більшої за дозволену під час авторизації) і натискання на кнопку «Створити».	Користувачу відображається головний екран зі списком платежів; у списку присутній невдалий платіж з специфічною червоною іконкою та двома піктограмами – перегляду (повторної спроби) та скасування. Користувачу відображається повідомлення про помилку від сервісу Coinbase.	Користувачу відобразився головний екран зі списком платежів; у списку був присутній невдалий платіж з специфічною червоною іконкою та двома піктограмами – перегляду (повторної спроби) та скасування. Користувачу відобразилось повідомлення про помилку від сервісу Coinbase (рис. 3.10).	Так

Відмітка про проходження: пройдений.

Рис. 3.10. Спроба редагувати накладну з незаповненими обов’язковими полями

Тестовий приклад: 4. розрахунок витрат.
Тест-вимоги, що перевіряються: 4. а.
Критерій проходження тесту: отримані результати збігаються з очікуваними.
Тест-план зі звітом наведений у табл. 3.6.

Таблиця 3.6

Тест-план «Розрахунок витрат»

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	Створити 4 сторонні регулярні платежі: перший у доларах з регулярністю в тиждень, другий у доларах з регулярністю в два тижні, третій у доларах з регулярністю в місяць, четвертий у гривнях з регулярністю в місяць. Сума оплати для кожного з них має бути рівна 1 у.о.	В списку регулярних платежів присутні 4 платежі з описаними параметрами.	В списку регулярних платежів присутні 4 платежі з описаними параметрами.	Так
2	Розрахувати витрати для валюти USD з періодичністю «Щомісяця»	Відображається попап-вікно з розрахованою сумою, рівною 7 USD.	Відобразилось попап-вікно з розрахованою сумою, рівною 7 USD (рис. 3.11).	Так

Відмітка про проходження: пройдений

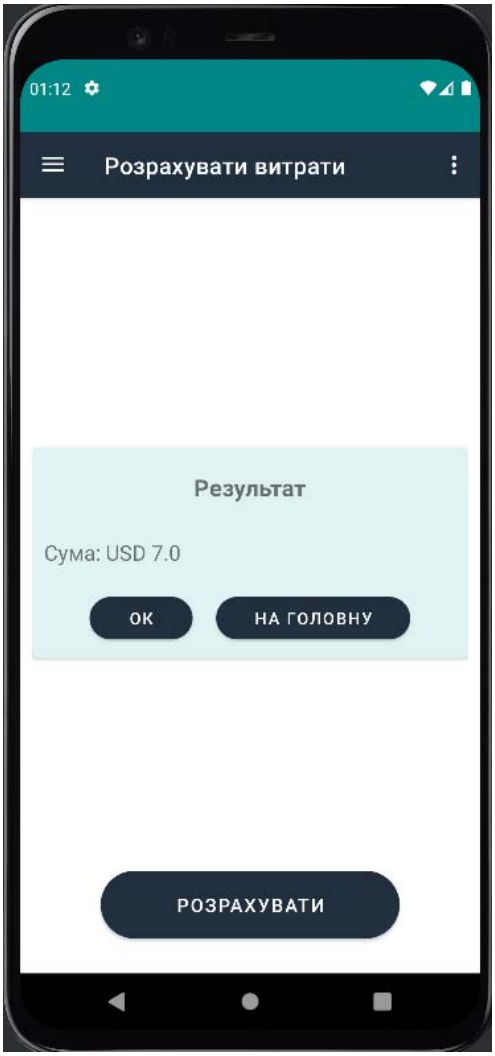

Рис. 3.11. Розраховані витрати

Тестових прикладів виконано: 5.
Тестових прикладів пройдено: 5.

3.5. Розгортання програмного продукту

Серверну частину застосунку у рамках розробки було розміщено на гостингу [24] pythonanywhere [25] за адресою https://notdatboi.pythonanywhere.com/. Так як, з міркувань безпеки, клієнтський застосунок може працювати тільки з сервером, розміщеним за наведеною адресою, додаткове розгортання серверної частини не потрібне.

Вимоги, необхідні для повного функціонування клієнтського застосунку:

- операційна система: Android.
- версія операційної системи: 8.0 та вище.
- об’єм оперативної пам’яті: 2 ГБ та більше.
- кількість ядер процесору: 4 ядра та більше.
- наявність вільного простору: 30 МБ та більше.
- доступ до інтернету: є.
- наявні застосунки у системі: будь-який веб браузер.
- дозвіл на встановлення застосунків за межами Google Play Store: дозволено.

Для розгортання клієнтського застосунку достатньо встановити відповідний файл Android Package (apk) [26]. Процес встановлення складається з таких кроків, які можуть незначно відрізнятись в залежності від версії ОС:

1. Завантажити застосунок, знайти його у файловій системі і натиснути на нього (рис. 3.12).
2. Підтвердити намір встановити застосунок (рис. 3.13).
3. Підтвердити намір встановити застосунок від невідомого розробника (натиснути кнопку «Install anyway») (рис. 3.14).
4. Дочекатись встановлення (рис. 3.15).
5. Обрати «Відкрити», коли застосунок буде встановлено (рис. 3.16).
6. Розпочати користування (рис. 3.17).

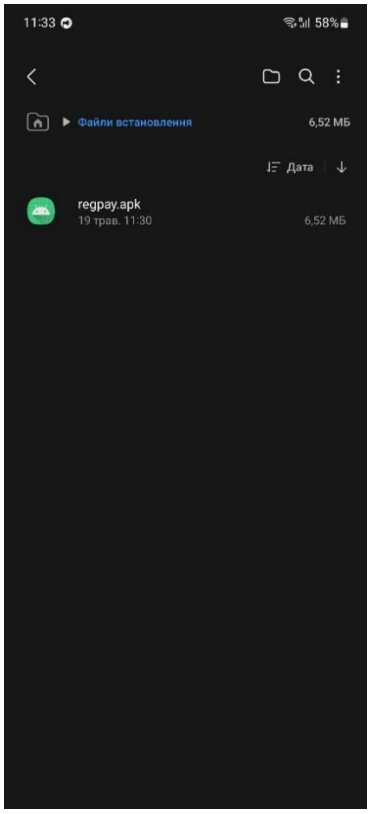

Рис. 3.12. Завантажений застосунок

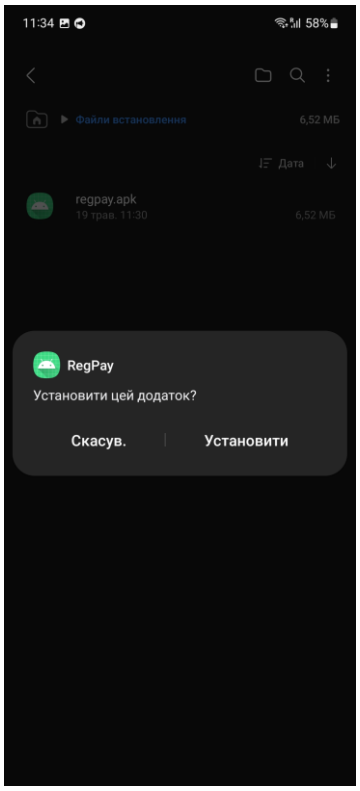

Рис. 3.13. Підтвердження встановлення застосунку

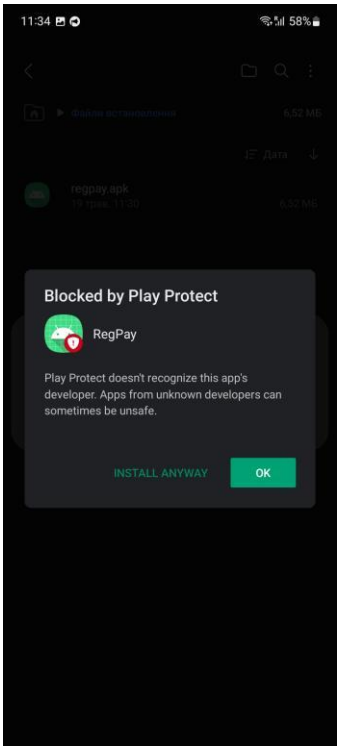

Рис. 3.14. Підтвердження встановлення застосунку від невідомого розробника

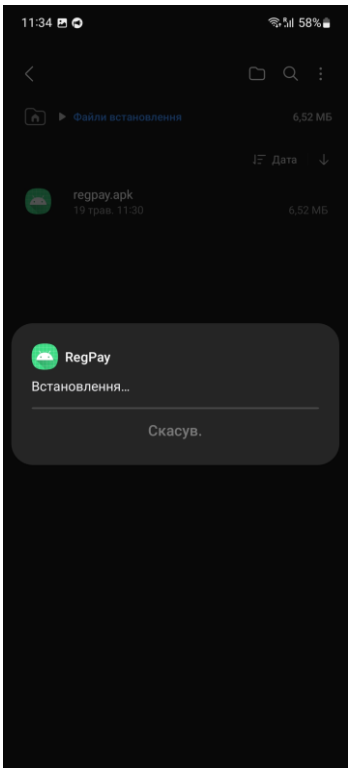

Рис. 3.15. Очікування встановлення

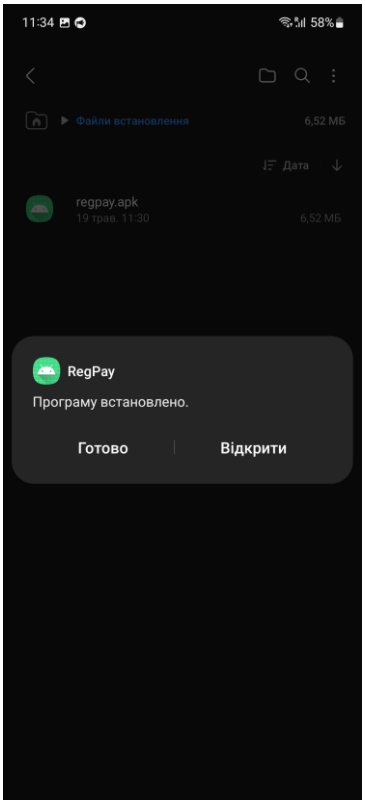

Рис. 3.16. Завершення встановлення

Рис. 3.17. Встановлений застосунок

Таким чином, в третьому розділі було вироблено математичну та логічну постановку задачі, здійснено концептуальне інфологічне проєктування [27],

спроєктовано логічну та фізичну моделі бази даних, а також проведено проєктування програмного забезпечення у вигляді діаграми класів та діаграми станів основних варіантів використання.

Також було проведено тестування програмної системи, наведено опис апаратних і програмних вимог для повноцінної роботи мобільного застосунку, а також описана процедура його розгортання на мобільному пристрої користувача.

ВИСНОВКИ

У рамках виконання дипломного проєкту було розроблено мобільний застосунок для управління регулярними платежами на базі клієнт-серверної архітектури [1] (модуль «Здійснення регулярних платежів»). Цей модуль призначений для створення регулярних платежів, контролю за створеними за допомогою модуля та сторонніми регулярними платежами, для підтримки списку регулярних платежів користувача у актуальному стані, а також для проведення розрахунків щомісячних, щотижневих та щодвотижневих обсягів фінансів, що витрачаються на такі платежі.

Під час розробки модуля була детально проаналізована предметна область із назвою «Управління регулярними платежами», під час чого було визначено проблеми у предметній області, проаналізовано існуючі аналоги, а також визначено актуальність та новизну набору функціональності, який було заплановано реалізувати в рамках розроблення модуля. Також були розроблені вимоги до системи, яка мала бути реалізована, словники даних для покращення розуміння замовником предметної області та термінології, використовуваної протягом проєкту. Заздалегідь було розроблено словник даних, на базі якого надалі було створено логічну та фізичну модель бази даних. Також за допомогою CASE-засобів [5] було продумано і створено архітектуру системи у вигляді UML-діаграм.

Для надійності і стабільності модуля його розробка велась за допомогою перевірених часом інструментів, бібліотек та фреймворків, до яких входять у тому числі інструменти Android Studio, PyCharm; набір бібліотек Android Jetpack (що включає бібліотеку Room); фреймворк Django. Ручне системне тестування модулю пройшло успішно, функціональність повністю відповідає вимогам.

Основними перевагами розробленого модулю є:

- легке додавання регулярних платежів зі сторонніх сервісів;
- можливість створювати нові регулярні платежі, не покидаючи додаток;
- відслідковування витрат одночасно за сторонніми та створеними у додатку регулярними платежами з визначеною періодичністю у визначеній валюті;
- відміна створених регулярних платежів не виходячи з додатку;
- можливість слідкувати за успішністю проведення регулярного платежу;
- зручний перегляд підписок.

Для розширення розроблюваного модулю планується додати функціональність сповіщення користувача про близький термін сплати платежу і про проблеми з оплатою створеної у застосунку підписки, а також інтеграцію з

більшою кількістю платіжних систем задля надання користувачу варіативності вибору системи, якою він хоче користуватись для оплати підписок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Client-Server Architecture for QA Engineers [Електронний ресурс] – Режим доступу: https://www.qamadness.com/knowledge-base/client-server-architecture-for-qa-engineers/
2. Akvelon Ukraine [Електронний ресурс] – Режим доступу: http://akvelon.com.ua/about-us-eng.aspx#company
3. Akvelon Inc. [Електронний ресурс] – Режим доступу: https://akvelon.com/
4. Business Dictionary [Електронний ресурс] – Режим доступу: https://web.archive.org/web/20200131001956/http://www.businessdictionary.com/
5. CASE [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/CASE
6. What is the BPMN 2.0 Standard? [Електронний ресурс] – Режим доступу: https://www.processmaker.com/blog/what-is-the-bpmn-2-0-standard/
7. Payment Services Directive [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Payment_Services_Directive#Revised_Directive_on_Payment_Services_(PSD2).
8. Get the transaction history from Android Pay via API? [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/44605295/get-the-transaction-history-from-android-pay-via-api
9. PayPal REST APIs [Електронний ресурс] – Режим доступу: https://developer.paypal.com/api/subscriptions/v1/
10. Developer Guides [Електронний ресурс] – Режим доступу: https://developer.authorize.net/api.html
11. Trim the Fat: How to Better Track and Manage Your Paid Subscriptions [Електронний ресурс] – Режим доступу: https://www.pcmag.com/how-to/track-and-manage-your-paid-subscriptions
12. Блог Пономарева: як називати application для смартфонів? [Електронний ресурс] – Режим доступу: https://www.bbc.com/ukrainian/blog-olexandr-ponomariv-38437307
13. Застосунок [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA
14. Криптовалюта [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D0%B0%D0%BB%D1%8E%D1%82%D0%B0
15. Що таке криптовалютний гаманець? [Електронний ресурс] – Режим доступу: https://academy.binance.com/uk/articles/crypto-wallet-types-explained

16. The great chain of being sure about things [Електронний ресурс] – Режим доступу: https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things
17. Android Toast [Електронний ресурс] – Режим доступу: http://tutorials.jenkov.com/android/toast.html
18. ОСОБА ФІЗИЧНА [Електронний ресурс] – Режим доступу: https://leksika.com.ua/15540812/legal/osoba_fizichna
19. Діаграма прецедентів — Use-case diagram [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html
20. What Is SQLite? [Електронний ресурс] – Режим доступу: https://www.sqlite.org/index.html
21. Save data in a local database using Room [Електронний ресурс] – Режим доступу: https://developer.android.com/training/data-storage/room
22. Design Patterns - MVC Pattern [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
23. Understanding the MVC pattern in Django [Електронний ресурс] – Режим доступу: https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f
24. Хостинг [Електронний ресурс] – Режим доступу: http://sum.in.ua/f/khostyngh
25. Host, run, and code Python in the cloud! [Електронний ресурс] – Режим доступу: https://www.pythonanywhere.com/
26. What Is an APK File and What Does It Do? Explained [Електронний ресурс] – Режим доступу: https://www.makeuseof.com/tag/what-is-apk-file/
27. Методичні рекомендації до виконання кваліфікаційних робіт для студентів спеціальностей 121 "Інженерія програмного забезпечення", 122 "Комп'ютерні науки", 126 "Інформаційні системи та технології" першого (бакалаврського) рівня [Електронний ресурс] / укл. Ю. Е. Парфьонов, І. О. Ушакова; ХНЕУ ім. С. Кузнеця. - Харків : ХНЕУ ім. С. Кузнеця, 2020. - 61 с.

ДОДАТКИ

Додаток А

Програмний код основних класів серверної частини застосунку

```
class LogInForm(forms.Form):
    pass


class UserManager:
    @staticmethod
    def create_user(username, email, password, is_admin=False):
        if not username or not email or not password or
User.objects.filter(username=username).count() != 0:
            return None
        user = User.objects.create_user(username=username, password=password,
email=email, is_staff=is_admin, is_superuser=is_admin)
        user.save()
        return user

    @staticmethod
    def log_in_user_custom_cred(request, username, password):
        user = authenticate(username=username, password=password)
        if user is None:
            user = authenticate(email=username, password=password)
        if user is None:
            return False
        login(request, user)
        return True

    @staticmethod
    def get_user_by_name(username):
        if username is None:
            return None
        query_set = User.objects.filter(username=username)
        if query_set.count() != 1:
            printl('ERROR BAD STUFF HAPPENING 2 OR MORE USERS WITH SAME USERNAME')
            return None  # should not happen as it's an unique field
        return query_set.get()


class Logger:
    path = '/home/notdatboi/regpay/regpay_app/logs/'
    filename = 'general.log'

    def log(self, data):
        if data is None:
            data = '<empty>'
        data = str(data)
        file = open(self.path + self.filename, 'a')
        dt = datetime.now() + timedelta(hours=3)
        file.write('[' + dt.strftime("%d-%b-%Y (%H:%M:%S.%f)") + '] ' + data + '\r\n')
        file.close()

GlobalLogger = Logger()

def printl(data):
    GlobalLogger.log(data)
```

```
class RPStatus(IntEnum):
    SUCCESS = 0
    ERROR = 1


# code style is messed here (and probably not only here), but I don't care
class RegularPaymentModel(models.Model):
    # name will be unique per user; sadly django does not normally support composite
PKs, or I'm looking in the wrong place
    name = models.CharField(max_length=50)
    picUrl = models.URLField(max_length=2000, blank=True)
    currency = models.IntegerField()
    amount = models.FloatField()
    regularity = models.IntegerField()
    service = models.IntegerField()
    destination = models.TextField(blank=True)
    status = models.IntegerField(default=int(RPStatus.SUCCESS))
    dateString = models.CharField(max_length=20, blank=True)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)

    class Meta:
        ordering = ['user']

    def __str__(self):
        return self.user.username + '/' + self.name

    def get_absolute_url(self):
        return reverse('model-detail-view', args=[str(self.id)])

    @staticmethod
    def get_all_rps():
        rps = RegularPaymentModel.objects.all()
        if rps.count() == 0:
            return None
        return rps

    @staticmethod
    def get_rps_by_date_string(date):
        rps = RegularPaymentModel.objects.filter(dateString=date)
        if rps.count() == 0:
            return None
        return rps

    @staticmethod
    def get_rps_by_user(user):
        if user is None:
            return None
        rps = RegularPaymentModel.objects.filter(user=user)
        if rps.count() == 0:
            return None
        return rps

    @staticmethod
    def get_rp_by_user_and_name(user, name):
        if user is None or name is None:
            return None
        rps = RegularPaymentModel.objects.filter(user=user, name=name)
        if rps.count() == 0:
```

```
            return None
        return rps.get()

    @staticmethod
    def delete_rp_by_user_and_name(user, name):
        rp = RegularPaymentModel.get_rp_by_user_and_name(user, name)
        if rp is not None:
            rp.delete()

    @staticmethod
    def get_rp_by_pk(pk):
        if pk is None:
            return None
        rps = RegularPaymentModel.objects.filter(pk=pk)
        if rps.count() == 0:
            return None
        return rps.get()

    @staticmethod
    def delete_rp_by_pk(pk):
        rp = RegularPaymentModel.get_rp_by_pk(pk)
        if rp is not None:
            rp.delete()



class UserDataModel(models.Model):
    key = models.CharField(max_length=100)
    data = models.TextField()
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)

    class Meta:
        ordering = ['key']
        constraints = [models.UniqueConstraint(fields=['key', 'user'],
name='unique_entry_key_per_user')]

    def __str__(self):
        return self.user.username + '/' + self.key

    def get_absolute_url(self):
        return reverse('model-detail-view', args=[str(self.id)])

    @staticmethod
    def get_data_by_user_and_key(user, key):
        obj = UserDataModel.get_by_user_and_key(user, key)
        if obj is None:
            return None
        return obj.data

    @staticmethod
    def get_by_user_and_key(user, key):
        if user is None:
            return None
        entries = UserDataModel.objects.filter(user=user, key=key)
        if entries.count() == 0:
            return None
        return entries.get()
```

```
class SystemSettingModel(models.Model):
    key = models.CharField(max_length=100, primary_key=True)
    value = models.TextField()

    class Meta:
        ordering = ['key']

    def __str__(self):
        return self.key

    def get_absolute_url(self):
        return reverse('model-detail-view', args=[str(self.key)])

    @staticmethod
    def get_value(key):
        entries = SystemSettingModel.objects.filter(key=key)
        if entries.count() == 0:
            return None
        return entries.get().value



class ClientError(IntEnum):
    INVALID_ACTION = 1
    SIGN_UP_FAILED_USER_ALREADY_EXISTS = 2
    SIGN_UP_FAILED_INVALID_DATA = 3
    SUCCESS = 4
    LOG_IN_FAILED_INVALID_DATA = 5
    LOG_IN_FAILED = 6
    INVALID_DATA = 7
    REG_PAYMENT_ALREADY_EXISTS = 8
    UNSUPPORTED_CURRENCY = 9
    TRANSACTION_FAILED = 10
    RESEND_WITH_TOKEN = 11
    SESSION_ON_OTHER_DEVICE = 12


class RPService(IntEnum):
    NONE = 0
    COINBASE = 1


class RPCurrency(IntEnum):
    USD = 0
    UAH = 1
    BTC = 2


class RPRegularity(IntEnum):
    MONTHLY = 0
    BIWEEKLY = 1
    WEEKLY = 2


class ClientAction(IntEnum):
    SIGN_UP = 1
    LOG_IN = 2
    GET_PAYMENTS = 3
    ADD_PAYMENT = 4
```

Продовження додатка А

```
    REMOVE_PAYMENT = 5
    COINBASE_AUTH = 6
    COINBASE_IS_AUTHORIZED = 7
    COINBASE_CREATE_PAYMENT = 8
    GET_SERVER_DATA = 9
    START_SESSION = 10
    END_SESSION = 11


class ServerAction:
    HANDLE_REGPAYMENTS = 1000


class ParamNames:
    action_param_name = 'action'
    device_id_param_name = 'device_id'
    payment_list_param_name = 'rps'
    username_param_name = 'username'
    email_param_name = 'email'
    password_param_name = 'password'
    error_param_name = 'error'
    retry_param_name = 'retry'
    error_desc_param_name = 'error_desc'
    name_param_name = 'name'
    date_param_name = 'date'
    pic_url_param_name = 'pic'
    currency_param_name = 'currency'
    amount_param_name = 'amount'
    regularity_param_name = 'regularity'
    service_param_name = 'service'
    status_param_name = 'status'
    coinbase_is_user_authorized_param_name = 'is_authorized'
    destination_param_name = "destination"
    coinbase_tfa_param_name = 'tfa'
    #completely internal for server
    coinbase_client_id_param_name = 'client_id'
    coinbase_client_secret_param_name = 'client_secret'
    #coinbase_user_authcode = 'code'
    coinbase_user_refresh_token_param_name = 'refresh_token'
    coinbase_user_account_param_name = 'account'
    coinbase_tfa_required = 'two_factor_required'


class MainView(View):
    template_name = 'log_in.html'

    def getCoinbaseAuthLink(self):
        params = urllib.parse.urlencode({'response_type' : 'code', 'client_id' :
SystemSettingModel.get_value(ParamNames.coinbase_client_id_param_name),
            'redirect_uri' : 'https://notdatboi.pythonanywhere.com/regpay_app'})
            #'redirect_uri' : 'urn:ietf:wg:oauth:2.0:oob'})
        url = 'https://www.coinbase.com/oauth/authorize?%s' % params;
        surl = str(url) +
'&scope=wallet:user:read,wallet:transactions:send,wallet:accounts:read,wallet:addresses:
read,wallet:buys:read&meta[send_limit_amount]=1&meta[send_limit_currency]=USD&meta[send_
limit_period]=day'
        return surl

    def authorizeOnCoinbase(self):
```

```
        surl = self.getCoinbaseAuthLink()
        printl(surl)
        rdr = redirect(surl)
        printl(rdr)
        return rdr

    def getAccessAndRefreshTokens(self, code):
        data = {'grant_type' : 'authorization_code',
            'code' : code,
            'client_id' :
SystemSettingModel.get_value(ParamNames.coinbase_client_id_param_name),
            'client_secret' :
SystemSettingModel.get_value(ParamNames.coinbase_client_secret_param_name),
            'redirect_uri' : 'https://notdatboi.pythonanywhere.com/regpay_app'}
            #'redirect_uri' : 'urn:ietf:wg:oauth:2.0:oob'}
        url = 'https://api.coinbase.com/oauth/token'
        resp = requests.post(url, data=data).json()
        #printl(resp.text)
        return resp['access_token'], resp['refresh_token']

    def updateAccessToken(self, refresh_token):
        data = {'grant_type' : 'refresh_token',
            'client_id' :
SystemSettingModel.get_value(ParamNames.coinbase_client_id_param_name),
            'client_secret' :
SystemSettingModel.get_value(ParamNames.coinbase_client_secret_param_name),
            'refresh_token' : refresh_token}
        url = 'https://api.coinbase.com/oauth/token'
        resp = requests.post(url, data=data).json()
        #printl(resp.text)
        if 'error' in resp:
            return None, None
        return resp['access_token'], resp['refresh_token']

    @staticmethod
    def currencyToText(rp_currency):
        if rp_currency == int(RPCurrency.USD):
            return 'USD'
        if rp_currency == int(RPCurrency.UAH):
            return 'UAH'
        if rp_currency == int(RPCurrency.BTC):
            return 'BTC'

    @staticmethod
    def getUserAccFromJSON(json):
        if 'error' in json or 'errors' in json:
            return None
        return json['data'][0]['id']

    def saveUserAccount(self, user):
        cud = UserDataModel.get_by_user_and_key(user=user,
key=ParamNames.coinbase_user_refresh_token_param_name)
        oldreftok = cud.data
        acctok, reftok = self.updateAccessToken(oldreftok)
        if reftok is None or acctok is None:
            cud.data = ''
            cud.save()
            return False
        cud.data = reftok
```

```
        cud.save()
        headers = {'Authorization': 'Bearer ' + acctok}
        url = 'https://api.coinbase.com/v2/accounts'
        resp = requests.get(url, headers=headers).json()
        printl(resp)
        useracc = MainView.getUserAccFromJSON(resp)
        if useracc is None:
            return False
        acc_cud = UserDataModel.get_by_user_and_key(user=user,
key=ParamNames.coinbase_user_account_param_name)
        if acc_cud is None:
            acc_cud = UserDataModel(user=user,
key=ParamNames.coinbase_user_account_param_name, data='')
        acc_cud.data = useracc
        acc_cud.save()
        return True

    def sendMoney(self, user, destination, amount, currency, tfa): # returns {error,
error_desc}
        cud = UserDataModel.get_by_user_and_key(user=user,
key=ParamNames.coinbase_user_refresh_token_param_name)
        oldreftok = cud.data
        acctok, reftok = self.updateAccessToken(oldreftok)
        if reftok is None or acctok is None:
            cud.data = ''
            cud.save()
            return ClientError.TRANSACTION_FAILED, None
        cud.data = reftok
        cud.save()
        acc = UserDataModel.get_data_by_user_and_key(user=user,
key=ParamNames.coinbase_user_account_param_name)
        if acc is None:
            return ClientError.TRANSACTION_FAILED, None
        json = {'type' : 'send',
            'to' : destination,
            'amount' : amount,
            'currency' : MainView.currencyToText(currency)}
        headers = None
        if tfa is None:
            headers = {'Authorization': 'Bearer ' + acctok}
        else:
            headers = {'Authorization': 'Bearer ' + acctok, 'CB-2FA-TOKEN': tfa}
        url = 'https://api.coinbase.com/v2/accounts/' + acc + '/transactions'
        printl(url)
        resp = requests.post(url, json=json, headers=headers).json()
        printl('here be response:')
        printl(resp)
        if 'error' in resp or 'errors' in resp:
            printl(resp['errors'][0]['id'])
            if 'errors' in resp and resp['errors'][0]['id'] ==
ParamNames.coinbase_tfa_required:
                return ClientError.RESEND_WITH_TOKEN, None
            return ClientError.TRANSACTION_FAILED, resp['errors'][0]['message']
        return ClientError.SUCCESS, None

    @staticmethod
    def processClientRequestError(err, desc=None):
        if desc is None:
            return JsonResponse({ParamNames.error_param_name: str(int(err))})
```

```
        return JsonResponse({ParamNames.error_param_name: str(int(err)),
ParamNames.error_desc_param_name: desc})

    def checkParamsInGet(self, request, params):
        for key in params:
            if (key not in request.GET or not request.GET[key]):
                return False
        return True

    def checkUser(self, request):
        success = UserManager.log_in_user_custom_cred(request,
request.GET[ParamNames.username_param_name],
request.GET[ParamNames.password_param_name])
        if not success:
            return None
        user = UserManager.get_user_by_name(request.GET[ParamNames.username_param_name])
        return user

    def processClientSignUp(self, request):
        if ParamNames.username_param_name not in request.GET or
ParamNames.email_param_name not in request.GET or ParamNames.password_param_name not in
request.GET \
            or not request.GET[ParamNames.username_param_name] or not
request.GET[ParamNames.email_param_name] or not
request.GET[ParamNames.password_param_name]:
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = UserManager.create_user(request.GET[ParamNames.username_param_name],
request.GET[ParamNames.email_param_name], request.GET[ParamNames.password_param_name])
        if user is None:
            return
MainView.processClientRequestError(ClientError.SIGN_UP_FAILED_USER_ALREADY_EXISTS)
        return MainView.processClientRequestError(ClientError.SUCCESS)

    def processClientLogIn(self, request):
        if ParamNames.username_param_name not in request.GET or
ParamNames.password_param_name not in request.GET \
            or not request.GET[ParamNames.username_param_name] or not
request.GET[ParamNames.password_param_name]:
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        success = UserManager.log_in_user_custom_cred(request,
request.GET[ParamNames.username_param_name],
request.GET[ParamNames.password_param_name])
        if not success:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        return MainView.processClientRequestError(ClientError.SUCCESS)

    # it's for 'importing' a payment; there will be a function
processClientCreatePayment for coinbase
    def processClientAddPayment(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
            ParamNames.password_param_name,
            ParamNames.name_param_name,
            ParamNames.pic_url_param_name,
            ParamNames.currency_param_name,
            ParamNames.amount_param_name,
            ParamNames.regularity_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
```

```
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        if RegularPaymentModel.get_rp_by_user_and_name(user,
request.GET[ParamNames.name_param_name]) is not None:
            return
MainView.processClientRequestError(ClientError.REG_PAYMENT_ALREADY_EXISTS)
        rpm = RegularPaymentModel(name=request.GET[ParamNames.name_param_name],
            picUrl=request.GET[ParamNames.pic_url_param_name],
            currency=int(request.GET[ParamNames.currency_param_name]),
            amount=float(request.GET[ParamNames.amount_param_name]),
            regularity=int(request.GET[ParamNames.regularity_param_name]),
            service=int(RPService.NONE),
            user=user)
        rpm.save()
        return MainView.processClientRequestError(ClientError.SUCCESS)

    def processClientRemovePayment(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name, ParamNames.name_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        RegularPaymentModel.delete_rp_by_user_and_name(user,
request.GET[ParamNames.name_param_name])
        return MainView.processClientRequestError(ClientError.SUCCESS)

    def processClientCoinbaseAuthorize(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        request.session[ParamNames.username_param_name] =
request.GET[ParamNames.username_param_name]
        request.session[ParamNames.password_param_name] =
request.GET[ParamNames.password_param_name]
        return self.authorizeOnCoinbase()

    def processClientCoinbaseCheckIfAuthorized(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        reftok = UserDataModel.get_data_by_user_and_key(user=user,
key=ParamNames.coinbase_user_refresh_token_param_name)
        is_auth = False
        if reftok is not None and len(reftok) > 0:
            is_auth = True
        return JsonResponse({ParamNames.error_param_name: str(int(ClientError.SUCCESS)),
ParamNames.coinbase_is_user_authorized_param_name: str(is_auth)})

    def processClientCoinbaseCreatePayment(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
            ParamNames.password_param_name,
            ParamNames.name_param_name,
            ParamNames.pic_url_param_name,
```

```
            ParamNames.currency_param_name,
            ParamNames.amount_param_name,
            ParamNames.regularity_param_name,
            ParamNames.destination_param_name,
            ParamNames.date_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        rpm = None
        date_str = request.GET[ParamNames.date_param_name]
        if self.checkParamsInGet(request, [ParamNames.retry_param_name]):
            rpm = RegularPaymentModel.get_rp_by_user_and_name(user,
request.GET[ParamNames.name_param_name])
        else:
            if RegularPaymentModel.get_rp_by_user_and_name(user,
request.GET[ParamNames.name_param_name]) is not None:
                return
MainView.processClientRequestError(ClientError.REG_PAYMENT_ALREADY_EXISTS)
        if rpm is None:
            rpm = RegularPaymentModel(name=request.GET[ParamNames.name_param_name],
                picUrl=request.GET[ParamNames.pic_url_param_name],
                destination=request.GET[ParamNames.destination_param_name],
                currency=int(request.GET[ParamNames.currency_param_name]),
                amount=float(request.GET[ParamNames.amount_param_name]),
                regularity=int(request.GET[ParamNames.regularity_param_name]),
                service=int(RPService.COINBASE),
                status=int(RPStatus.SUCCESS),
                dateString=date_str,
                user=user)
        tfa = None
        if self.checkParamsInGet(request, [ParamNames.coinbase_tfa_param_name]):
            tfa = request.GET[ParamNames.coinbase_tfa_param_name]
        error, error_desc = self.sendMoney(user, rpm.destination, rpm.amount,
rpm.currency, tfa)
        printl(error)
        if error is not ClientError.SUCCESS:
            rpm.status = int(RPStatus.ERROR)
        if error is not ClientError.RESEND_WITH_TOKEN:
            rpm.save() # don't save in case of token resend response, because client
will call this method again with token supplied - we'll save it then
        return MainView.processClientRequestError(error, error_desc)

    def processClientGetServerData(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        rps = RegularPaymentModel.get_rps_by_user(user)
        if rps is not None:
            rp_data_list = list()
            for rp in rps:
                rp_data = {ParamNames.name_param_name: rp.name,
                    ParamNames.pic_url_param_name: rp.picUrl,
                    ParamNames.currency_param_name: rp.currency,
                    ParamNames.amount_param_name: rp.amount,
                    ParamNames.regularity_param_name: rp.regularity,
```

```
                    ParamNames.service_param_name: rp.service,
                    ParamNames.status_param_name: rp.status,
                    ParamNames.destination_param_name: rp.destination}
                rp_data_list.append(rp_data)
            return JsonResponse({ParamNames.error_param_name:
str(int(ClientError.SUCCESS)), ParamNames.payment_list_param_name: rp_data_list})
        return MainView.processClientRequestError(ClientError.SUCCESS)

    def processClientStartSession(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name, ParamNames.device_id_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        dev_id_entry = UserDataModel.get_by_user_and_key(user=user,
key=ParamNames.device_id_param_name)
        if dev_id_entry is not None and dev_id_entry.data ==
request.GET[ParamNames.device_id_param_name]: # app probably crashed and didn't end
session; allow it to continue this one
            return MainView.processClientRequestError(ClientError.SUCCESS)
        if dev_id_entry is None:
            UserDataModel(user=user, key=ParamNames.device_id_param_name,
data=request.GET[ParamNames.device_id_param_name]).save()
            return MainView.processClientRequestError(ClientError.SUCCESS)
        return MainView.processClientRequestError(ClientError.SESSION_ON_OTHER_DEVICE)

    def processClientEndSession(self, request):
        if not self.checkParamsInGet(request, [ParamNames.username_param_name,
ParamNames.password_param_name]):
            return MainView.processClientRequestError(ClientError.INVALID_DATA)
        user = self.checkUser(request)
        if user is None:
            return MainView.processClientRequestError(ClientError.LOG_IN_FAILED)
        dev_id_entry = UserDataModel.get_by_user_and_key(user=user,
key=ParamNames.device_id_param_name)
        if dev_id_entry is not None:
            dev_id_entry.delete()
        return MainView.processClientRequestError(ClientError.SUCCESS)

    def processClientRequest(self, request):
        if ParamNames.action_param_name not in request.GET:
            return MainView.processClientRequestError(ClientError.INVALID_ACTION)
        action = 0
        try:
            action = int(request.GET[ParamNames.action_param_name])
        except:
            return MainView.processClientRequestError(ClientError.INVALID_ACTION)
        if action == ClientAction.SIGN_UP:
            return self.processClientSignUp(request)
        elif action == ClientAction.LOG_IN:
            return self.processClientLogIn(request)
        elif action == ClientAction.ADD_PAYMENT:
            return self.processClientAddPayment(request)
        elif action == ClientAction.REMOVE_PAYMENT:
            return self.processClientRemovePayment(request)
        elif action == ClientAction.COINBASE_AUTH:
            return self.processClientCoinbaseAuthorize(request)
        elif action == ClientAction.COINBASE_IS_AUTHORIZED:
```

```
            return self.processClientCoinbaseCheckIfAuthorized(request)
        elif action == ClientAction.COINBASE_CREATE_PAYMENT:
            return self.processClientCoinbaseCreatePayment(request)
        elif action == ClientAction.GET_SERVER_DATA:
            return self.processClientGetServerData(request)
        elif action == ClientAction.START_SESSION:
            return self.processClientStartSession(request)
        elif action == ClientAction.END_SESSION:
            return self.processClientEndSession(request)
        else:
            return MainView.processClientRequestError(ClientError.INVALID_ACTION)

    def get(self, request, *args, **kwargs):
        printl(request)
        if 'code' in request.GET:
            printl('code in get!')
            printl(request.session[ParamNames.username_param_name])
            printl(request.GET)
            success = UserManager.log_in_user_custom_cred(request,
request.session[ParamNames.username_param_name],
request.session[ParamNames.password_param_name])
            if success:
                acctok, reftok = self.getAccessAndRefreshTokens(request.GET['code'])
                reftok_entry =
UserDataModel(key=ParamNames.coinbase_user_refresh_token_param_name,
user=UserManager.get_user_by_name(request.session[ParamNames.username_param_name]),
data=reftok)
                reftok_entry.save()
                user =
UserManager.get_user_by_name(request.session[ParamNames.username_param_name])
                if self.saveUserAccount(user):
                    printl('successfully got user!')
                else:
                    printl('user get was not successful')
            return render(request, 'compl_auth.html', context={})
        action = 0
        try:
            action = int(request.GET[ParamNames.action_param_name])
        except:
            pass
        printl(action)
        if ParamNames.action_param_name in request.GET and action ==
ServerAction.HANDLE_REGPAYMENTS:
            self.conductRegularPayments()
            return MainView.processClientRequestError(ClientError.SUCCESS)
        return self.processClientRequest(request)

    def post(self, request, *args, **kwargs):
        if 'code' not in request.session or request.session['code'] is None:
            return self.authorizeOnCoinbase()
        else:
            resp = self.getAccessAndRefreshTokens(request.session['code'])
            #request.session['code'] = None
            return render(request, self.template_name, context={'form': LogInForm()})

    def conductRegularPayments(self):
        rps = RegularPaymentModel.get_all_rps()
        printl('staring conducting RPs')
        today = date.today()
```

Закінчення додатка А

```
        printl('today: ' + str(today))
        if rps is not None:
            for rp in rps:
                printl('rp: ' + rp.name)
                if rp.dateString is not None and len(rp.dateString) != 0 and rp.service
== int(RPService.COINBASE):
                    rp_date = datetime.strptime(rp.dateString, "%d/%m/%Y").date()
                    printl('rp_date: ' + str(rp_date))
                    if rp_date == today: # if it is equal, then we already updated it
before
                        continue
                    rewrite = False
                    if rp.regularity == int(RPRegularity.MONTHLY) and rp_date.day ==
today.day:
                        printl('do the job mon')
                        rewrite = True
                    if rp.regularity == int(RPRegularity.BIWEEKLY) and (today -
rp_date).days == 14:
                        printl('do the job biweek')
                        rewrite = True
                    if rp.regularity == int(RPRegularity.WEEKLY) and (today -
rp_date).days == 7:
                        printl('do the job week')
                        rewrite = True
                    if rewrite:
                        err, desc = self.sendMoney(rp.user, rp.destination, rp.amount,
rp.currency, None)
                        if err == ClientError.SUCCESS:
                            rp.status = int(RPStatus.SUCCESS)
                        else:
                            rp.status = int(RPStatus.ERROR)
                        printl(err)
                        printl(desc)
                        rp.dateString = today.strftime('%d/%m/%Y')
                        rp.save()


class CompletedAuthView(View):
    template_name = 'compl_auth.html'

    def get(self, request, *args, **kwargs):
        return render(request, self.template_name, context={})

    def post(self, request, *args, **kwargs):
        return render(request, self.template_name, context={})
```

Додаток Б

Програмний код основних класів клієнтської частини застосунку

```
class SignUpActivity : AppCompatActivity()
{
    override fun onCreate(savedInstanceState: Bundle?)
    {
        // todo: REMOVE NEXT 2 LINES ASAP \/
        val policy = StrictMode.ThreadPolicy.Builder().permitAll().build()
        StrictMode.setThreadPolicy(policy)

        AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);

        setTitle(R.string.sign_up_layout_name)

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_signup)

        // check if we have been authorized before
        // todo: should be rewritten when password change is supported
        val db = AppDatabase.getInstance(applicationContext)
        var shouldProceed = false
        runBlocking {
            launch(Dispatchers.IO) {
                val matchingSetting =
db.getSystemSettingDao().getSetting(Settings.LAST_USER)
                if (matchingSetting != null)
                    shouldProceed = true
            }
        }
        if (shouldProceed)
        {
            val intent = Intent(this, NavDrawerTestActivity::class.java)
            startActivity(intent)
            finish()
        }

        findViewById<Button>(R.id.signUpBtn).setOnClickListener {
            val db = AppDatabase.getInstance(applicationContext)
            val user = User(findViewById<EditText>(R.id.loginVal).text.toString(),
                md5(findViewById<EditText>(R.id.passwordVal).text.toString()))
            val err = ServerConnectionHandler.sendSignUpRequest(user.username,
findViewById<EditText>(R.id.emailVal).text.toString(), user.passwordHash)
            ServerConnectionHandler.toastIfError(applicationContext, err)
            if (err == ServerConstants.ErrorValue.SUCCESS)
            {
                println("success!")
                runBlocking {
                    launch(Dispatchers.IO) {
                        db.getUserDao().insertUser(user)

db.getSystemSettingDao().insertSetting(SystemSetting(Settings.LAST_USER, user.username))
                        db.getUserDao().getUsers().forEach {
                            println(it.username + "|" + it.passwordHash)
                        }
                    }
                }
                val intent = Intent(this, LogInActivity::class.java)

                startActivity(intent)
```

```
                finish()
            }
            else
            {
                println("error!")
            }
        }
        findViewById<Button>(R.id.logInBtn).setOnClickListener {
            val intent = Intent(this, LogInActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}


object ServerConstants
{
    // basic
    const val ACTION_KEY = "action"
    const val ERROR_KEY = "error"
    const val ERROR_DESC_KEY = "error_desc"
    const val DEVICE_ID_KEY = "device_id"
    const val PAYMENT_LIST_KEY = "rps"
    enum class ActionValue(val value: String)
    {
        SIGN_UP("1"),
        LOG_IN("2"),
        GET_PAYMENTS("3"),
        ADD_PAYMENT("4"),
        REMOVE_PAYMENT("5"),
        COINBASE_AUTH("6"),
        COINBASE_IS_AUTHORIZED("7"),
        COINBASE_CREATE_PAYMENT("8"),
        GET_SERVER_DATA("9"),
        START_SESSION("10"),
        END_SESSION("11");

        companion object
        {
            fun fromString(value: String) = ActionValue.values().first { it.value ==
value }
        }
    }
    enum class ErrorValue(val value: Int)
    {
        NO_RESP_FROM_SERVER(0), // used only on client
        INVALID_ACTION(1),
        SIGN_UP_FAILED_USER_ALREADY_EXISTS(2),
        SIGN_UP_FAILED_INVALID_DATA(3),
        SUCCESS(4),
        LOG_IN_FAILED_INVALID_DATA(5),
        LOG_IN_FAILED(6),
        INVALID_DATA(7),
        REG_PAYMENT_ALREADY_EXISTS(8),
        UNSUPPORTED_CURRENCY(9),
        TRANSACTION_FAILED(10),
        RESEND_WITH_TOKEN(11),
```

```
        SESSION_ON_OTHER_DEVICE(12);

        companion object
        {
            fun fromInt(value: Int) = ErrorValue.values().first { it.value == value }
            fun toUiString(context: Context, err: ErrorValue): String =
context.resources.getStringArray(R.array.errors)[err.value]
        }
    }
    // user
    const val USERNAME_KEY = "username"
    const val EMAIL_KEY = "email"
    const val PASSWORD_KEY = "password"
    // payment
    const val NAME_KEY = "name"
    const val PIC_URL_KEY = "pic"
    const val CURRENCY_KEY = "currency"
    const val AMOUNT_KEY = "amount"
    const val REGULARITY_KEY = "regularity"
    const val DESTINATION_KEY = "destination"
    const val SERVICE_KEY = "service"
    const val STATUS_KEY = "status"
    const val RETRY_KEY = "retry"
    const val DATE_KEY = "date"
    // coinbase
    const val COINBASE_AUTH_LINK = "auth_link"
    const val COINBASE_IS_USER_AUTHORIZED = "is_authorized"
    const val COINBASE_TFA_KEY = "tfa"
}

class ServerConnectionHandler
{
    companion object
    {
        private const val ServerUrl = "https://notdatboi.pythonanywhere.com/regpay_app";

        fun toastIfError(context: Context, err: ServerConstants.ErrorValue,
additionalErrorDesc: String? = null)
        {
            var text = ServerConstants.ErrorValue.toUiString(context, err)
            if (err == ServerConstants.ErrorValue.TRANSACTION_FAILED &&
additionalErrorDesc != null)
            {
                text += " " + context.getString(R.string.server_details) + " " +
additionalErrorDesc
            }
            if (err != ServerConstants.ErrorValue.SUCCESS)
                Toast.makeText(context, text, Toast.LENGTH_LONG).show()
        }

        fun sendSignUpRequest(username: String, email: String?, passwordHash: String):
ServerConstants.ErrorValue
        {
            val resp = sendGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.SIGN_UP.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.EMAIL_KEY to (email ?: "example@mail.com"),
                ServerConstants.PASSWORD_KEY to passwordHash))
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
```

```
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY));
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }

        fun sendLogInRequest(username: String, passwordHash: String):
ServerConstants.ErrorValue
        {
            val resp = sendGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.LOG_IN.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash))
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY));
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }

        fun sendAddRegularPaymentRequest(username: String, passwordHash: String, rpName:
String, pic: String, currency: RPCurrency, amount: Double, regularity: RPRegularity):
ServerConstants.ErrorValue
        {
            val resp = sendGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.ADD_PAYMENT.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash,
                ServerConstants.NAME_KEY to rpName,
                ServerConstants.PIC_URL_KEY to pic,
                ServerConstants.CURRENCY_KEY to currency.value.toString(),
                ServerConstants.AMOUNT_KEY to amount.toString(),
                ServerConstants.REGULARITY_KEY to regularity.value.toString()))
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY));
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }

        fun sendRemoveRegularPaymentRequest(username: String, passwordHash: String,
rpName: String): ServerConstants.ErrorValue
        {
            val resp = sendGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.REMOVE_PAYMENT.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash,
                ServerConstants.NAME_KEY to rpName))
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY));
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }

        fun composeCoinbaseAuthRequest(username: String, passwordHash: String): String
        {
            return composeGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.COINBASE_AUTH.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash))
        }
```

```
        fun sendCheckIfAuthorizedOnCoinbase(username: String, passwordHash: String):
Pair<ServerConstants.ErrorValue, JSONObject?>
        {
            val resp = sendGet(mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.COINBASE_IS_AUTHORIZED.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash))
            var err = ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
            {
                err =
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY))
                if (err == ServerConstants.ErrorValue.SUCCESS)
                    return Pair(err, resp)
            }
            return Pair(err, null)
        }

        fun sendCreateRegularPaymentRequest(username: String, passwordHash: String,
rpName: String, pic: String, currency: RPCurrency, amount: Double, regularity:
RPRegularity, destination: String, dateString: String, retry: Boolean = false, tfa:
String? = null): Pair<ServerConstants.ErrorValue, String?>
        {
            val data = mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.COINBASE_CREATE_PAYMENT.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash,
                ServerConstants.NAME_KEY to rpName,
                ServerConstants.PIC_URL_KEY to pic,
                ServerConstants.CURRENCY_KEY to currency.value.toString(),
                ServerConstants.AMOUNT_KEY to amount.toString(),
                ServerConstants.REGULARITY_KEY to regularity.value.toString(),
                ServerConstants.DESTINATION_KEY to destination,
                ServerConstants.DATE_KEY to dateString)
            if (tfa != null)
                data[ServerConstants.COINBASE_TFA_KEY] = tfa
            if (retry)
                data[ServerConstants.RETRY_KEY] = "true"
            val resp = sendGet(data)
            var desc: String? = null
            if (resp != null && resp.has(ServerConstants.ERROR_DESC_KEY))
                desc = resp.getString(ServerConstants.ERROR_DESC_KEY)
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
                return
Pair(ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY)), desc)
            return Pair(ServerConstants.ErrorValue.NO_RESP_FROM_SERVER, desc)
        }

        fun sendGetServerData(username: String, passwordHash: String):
Pair<ServerConstants.ErrorValue, ArrayList<RegularPayment>?>
        {
            val data = mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.GET_SERVER_DATA.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash)
            val resp = sendGet(data)
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
            {
                var serverData: ArrayList<RegularPayment>? = null
```

Продовження додатка Б

```
                if (resp.has(ServerConstants.PAYMENT_LIST_KEY))
                {
                    val array = resp.getJSONArray(ServerConstants.PAYMENT_LIST_KEY)
                    for (i in 0 until array.length())
                    {
                        val item = array.getJSONObject(i)
                        val rp =
RegularPayment(item.getString(ServerConstants.NAME_KEY),
                            item.getString(ServerConstants.PIC_URL_KEY),
                            item.getInt(ServerConstants.CURRENCY_KEY),
                            item.getDouble(ServerConstants.AMOUNT_KEY),
                            item.getInt(ServerConstants.REGULARITY_KEY),
                            item.getInt(ServerConstants.SERVICE_KEY),
                            item.getInt(ServerConstants.STATUS_KEY),
                            item.getString(ServerConstants.DESTINATION_KEY).ifEmpty {
null })
                        if (serverData == null)
                            serverData = arrayListOf()
                        serverData.add(rp)
                    }
                }
                return
Pair(ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY)),
serverData)
            }
            return Pair(ServerConstants.ErrorValue.NO_RESP_FROM_SERVER, null)
        }

        fun sendStartSession(username: String, passwordHash: String, deviceId: String):
ServerConstants.ErrorValue
        {
            val data = mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.START_SESSION.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash,
                ServerConstants.DEVICE_ID_KEY to deviceId)
            val resp = sendGet(data)
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
            {
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY))
            }
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }

        fun sendEndSession(username: String, passwordHash: String):
ServerConstants.ErrorValue
        {
            val data = mutableMapOf(ServerConstants.ACTION_KEY to
ServerConstants.ActionValue.END_SESSION.value,
                ServerConstants.USERNAME_KEY to username,
                ServerConstants.PASSWORD_KEY to passwordHash)
            val resp = sendGet(data)
            if (resp != null && resp.has(ServerConstants.ERROR_KEY))
                return
ServerConstants.ErrorValue.fromInt(resp.getInt(ServerConstants.ERROR_KEY))
            return ServerConstants.ErrorValue.NO_RESP_FROM_SERVER
        }
```

Продовження додатка Б

```
        fun composeGet(data: MutableMap<String, String>) : String
        {
            var currentUrl = ServerUrl;
            var isFirst = true
            data.forEach {
                currentUrl += (if (isFirst) "?" else "&") + it.key + "=" + it.value
                isFirst = false
            }
            return currentUrl
        }

        fun sendGet(data: MutableMap<String, String>): JSONObject?
        {
            val url = URL(composeGet(data));
            with(url.openConnection() as HttpURLConnection) {
                requestMethod = "GET"  // optional default is GET

                println("\nSent 'GET' request to URL : $url; Response Code :
$responseCode");

                var fullText = ""
                if (responseCode == 200)
                {
                    inputStream.bufferedReader().use {
                        it.lines().forEach { line ->
                            fullText += line
                        }
                    }
                }
                println(fullText)
                try {
                    return JSONObject(fullText)
                }
                catch (e: JSONException){
                    e.printStackTrace()
                }
            }
            return null
        }
    }
}


class RegularPaymentModel(application: Application) : AndroidViewModel(application)
{
    var dataSource: MutableLiveData<ArrayList<RegularPayment>> = MutableLiveData();

    fun getCurrentUserCreds(): Pair<String, String>? =
GetCurrentUserCreds(getApplication())

    init
    {
        runBlocking {
            launch(Dispatchers.IO) {

AppDatabase.getInstance(application.applicationContext).getRegularPaymentDao().deleteAll
RegularPayments()
            }
        }
```

```
        val creds = getCurrentUserCreds()
        if (creds != null)
        {
            val respAndData = ServerConnectionHandler.sendGetServerData(creds.first,
creds.second)
            ServerConnectionHandler.toastIfError(application.applicationContext,
respAndData.first)
            if (respAndData.first == ServerConstants.ErrorValue.SUCCESS &&
respAndData.second != null)
            {
                runBlocking {
                    launch(Dispatchers.IO) {
                        val db = AppDatabase.getInstance(application.applicationContext)
                        respAndData.second!!.forEach {
                            db.getRegularPaymentDao().insertRegularPayment(it)
                        }
                    }
                }
            }
        }
        var data = ArrayList<RegularPayment>()
        runBlocking {
            launch(Dispatchers.IO) {
                data =
AppDatabase.getInstance(application.applicationContext).getRegularPaymentDao().getRegula
rPayments().toCollection(ArrayList())
            }
        }
        dataSource.value = data
    }

    fun getData() = dataSource;
}


class RegularPaymentAdapter
(
    private val context: Context,
    private val navDrawerActivity: NavDrawerTestActivity
) : RecyclerView.Adapter<RegularPaymentAdapter.RegularPaymentHolder>()
{
    class RegularPaymentHolder(val rowView: View) : RecyclerView.ViewHolder(rowView)
    {
        var active = false;
        fun bind(regularPayment: RegularPayment, navDrawerActivity:
NavDrawerTestActivity, position: Int)
        {
            with(rowView)
            {
                val picUrlSuccess = "https://www.tspa.org/wp-
content/uploads/2021/09/coinbase.png"
                val picUrlError = "https://icon-library.com/images/error-icon-
transparent/error-icon-transparent-5.jpg"
                val nameView = rowView.findViewById(R.id.serviceName) as TextView
                val priceView = rowView.findViewById(R.id.servicePrice) as TextView
                val currencyView = rowView.findViewById(R.id.currency) as TextView
                val iconView = rowView.findViewById(R.id.serviceIcon) as ImageView
                val editBtn = rowView.findViewById(R.id.editRPButton) as ImageButton
```

Продовження додатка Б

```
                nameView.text = if (regularPayment.status == RPStatus.ERROR.value)
regularPayment.name + " " + navDrawerActivity.getString(R.string.payment_error) else
regularPayment.name;
                priceView.text = regularPayment.amount.toString();
                currencyView.text = RPCurrency.toUiString(context,
RPCurrency.fromInt(regularPayment.currency))
                if (regularPayment.service == RPService.COINBASE.value)
                {
                    if (regularPayment.status == RPStatus.SUCCESS.value)
                        editBtn.visibility = View.INVISIBLE
                    else
                        editBtn.setImageResource(R.drawable.ic_look)
                }
                if (regularPayment.service == RPService.COINBASE.value)
                {
                    if (regularPayment.status == RPStatus.SUCCESS.value)

Picasso.with(context).load(picUrlSuccess).placeholder(R.mipmap.ic_launcher).into(iconVie
w)
                    else

Picasso.with(context).load(picUrlError).placeholder(R.mipmap.ic_launcher).into(iconView)


rowView.findViewById<ImageButton>(R.id.deleteRPButton).setOnClickListener {
                        navDrawerActivity.spawnRPDataRemovalPopup(position)
                    }

rowView.findViewById<ImageButton>(R.id.editRPButton).setOnClickListener {
                        //navDrawerActivity.startEdit(position)
                        val item = navDrawerActivity.adapter!!.getItem(position) as
RegularPayment
                        val bundle = bundleOf("RegularPayment" to item, "index" to
position)

navDrawerActivity.findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navi
gate(R.id.nav_create, bundle)
                    }
                }
                else
                {

Picasso.with(context).load(regularPayment.picUrl).placeholder(R.mipmap.ic_launcher).into
(iconView);

rowView.findViewById<ImageButton>(R.id.deleteRPButton).setOnClickListener {
                        navDrawerActivity.spawnRPDataRemovalPopup(position)
                    }

rowView.findViewById<ImageButton>(R.id.editRPButton).setOnClickListener {
                        //navDrawerActivity.startEdit(position)
                        val item = navDrawerActivity.adapter!!.getItem(position) as
RegularPayment
                        val bundle = bundleOf("RegularPayment" to item, "index" to
position)

navDrawerActivity.findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navi
gate(R.id.nav_edit, bundle)
                    }
```

Продовження додатка Б

```
                }
                /*rowView.setOnClickListener {
                    active = true;
                    it.showContextMenu();
                }
                rowView.setOnLongClickListener {
                    // avoiding context menu call on long click
                    //parentMainActivity.onLongConfPartClick(position, confPart);
                    return@setOnLongClickListener true;
                };*/
            }
        }
    }

    private var dataSource: ArrayList<RegularPayment> = ArrayList();
    private val inflater: LayoutInflater =
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater;
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RegularPaymentAdapter.RegularPaymentHolder
    {
        return
RegularPaymentHolder(LayoutInflater.from(parent.context).inflate(R.layout.list_item_regu
lar_payment, parent, false))
    }

    override fun onBindViewHolder(holder: RegularPaymentHolder, position: Int)
    {
        holder.bind(dataSource[position], navDrawerActivity, position);
    }

    override fun getItemCount(): Int
    {
        return dataSource.size;
    }

    fun updateRegularPayments(payments: ArrayList<RegularPayment>)
    {
        this.dataSource = payments;
        notifyDataSetChanged();
    }

    /*override*/ fun getItem(position: Int): Any
    {
        return dataSource[position];
    }

    override fun getItemId(position: Int): Long
    {
        return position.toLong();
    }

    fun removeByIndex(position: Int)
    {
        dataSource.removeAt(position);
        notifyDataSetChanged();
    }

    fun editByIndex(position: Int, payment: RegularPayment)
    {
```

```
        dataSource[position] = payment;
        notifyDataSetChanged();
    }

    fun add(payment: RegularPayment)
    {
        dataSource.add(payment);
        notifyDataSetChanged();
    }
}


class NavDrawerTestActivity : AppCompatActivity() {

    private lateinit var appBarConfiguration: AppBarConfiguration
    private lateinit var binding: ActivityNavDrawerTestBinding
    private lateinit var _adapter: RegularPaymentAdapter;
    val adapter get() = if (this::_adapter.isInitialized) _adapter else null

    @SuppressLint("HardwareIds")
    fun getUniqueDeviceId(): String
    {
        val uniqueId: String =
android.provider.Settings.Secure.getString(getContentResolver(),
android.provider.Settings.Secure.ANDROID_ID)
        return uniqueId
    }

    private var dontEndSession: Boolean = false

    private fun startSession()
    {
        println("starting RegPay session")
        val creds = getCurrentUserCreds()
        if (creds != null)
        {
            val err = ServerConnectionHandler.sendStartSession(creds.first,
creds.second, getUniqueDeviceId())
            ServerConnectionHandler.toastIfError(applicationContext, err)
            if (err == ServerConstants.ErrorValue.SESSION_ON_OTHER_DEVICE)
            {
                dontEndSession = true
                finish()
            }
        }
    }

    private fun endSession()
    {
        if (dontEndSession)
            return
        println("ending RegPay session")
        val creds = getCurrentUserCreds()
        if (creds != null)
        {
            val err = ServerConnectionHandler.sendEndSession(creds.first, creds.second)
            ServerConnectionHandler.toastIfError(applicationContext, err)
        }
    }
```

Продовження додатка Б

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        _adapter = RegularPaymentAdapter(this, this)

        binding = ActivityNavDrawerTestBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setSupportActionBar(binding.appBarNavDrawerTest.toolbar)

        startSession()

        val drawerLayout: DrawerLayout = binding.drawerLayout
        val navView: NavigationView = binding.navView
        val navController =
findNavController(R.id.nav_host_fragment_content_nav_drawer_test)

        binding.navView
        appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.nav_list, R.id.nav_import, R.id.nav_create, R.id.nav_calc
            ), drawerLayout
        )
        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)

        val creds = getCurrentUserCreds()
        if (creds != null)
        {
            val headerView = navView.getHeaderView(0)
            headerView.findViewById<TextView>(R.id.usernameText).text = creds.first
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        // Inflate the menu; this adds items to the action bar if it is present.
        menuInflater.inflate(R.menu.nav_drawer_test, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
    {
        if (item.itemId == R.id.logOutAction)
        {
            runBlocking {
                launch(Dispatchers.IO) {
                    val db = AppDatabase.getInstance(applicationContext)
                    db.getSystemSettingDao().deleteSettingByName(Settings.LAST_USER)
                }
            }
            val intent = Intent(this, SignUpActivity::class.java)
            startActivity(intent)
            finish()
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onSupportNavigateUp(): Boolean {
```

Продовження додатка Б

```
        val navController =
findNavController(R.id.nav_host_fragment_content_nav_drawer_test)
        return navController.navigateUp(appBarConfiguration) ||
super.onSupportNavigateUp()
    }

    fun getCurrentUserCreds(): Pair<String, String>? =
GetCurrentUserCreds(applicationContext)

    fun spawnPopup(headerId: Int, textId: Int? = null, textString: String? = null,
leftBtnTxtId: Int? = null, rightBtnTxtId: Int? = null): Pair<View, PopupWindow>
    {
        val popupView = layoutInflater.inflate(R.layout.confirmation_popup, null)
        val popupWindow = PopupWindow(popupView, LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT, true)
        popupWindow.showAtLocation(binding.navView, Gravity.CENTER, 0, 0)
        popupView.findViewById<TextView>(R.id.titleText).text = getString(headerId)
        if (textId != null)
            popupView.findViewById<TextView>(R.id.messageText).text = getString(textId)
        if (textString != null)
            popupView.findViewById<TextView>(R.id.messageText).text = textString
        if (leftBtnTxtId != null)
            popupView.findViewById<Button>(R.id.yesButton).text =
getString(leftBtnTxtId)
        if (rightBtnTxtId != null)
            popupView.findViewById<Button>(R.id.noButton).text =
getString(rightBtnTxtId)
        return Pair(popupView, popupWindow)
    }

    fun spawnCalcSpendingPopup(sum: Double, currency: String) {
        val text = getString(R.string.calc_result_text) + " " + currency + " " +
sum.toString()
        val popupViewAndWindow = spawnPopup(
            R.string.calc_result_header,
            null,
            text,
            R.string.option_back_to_calc,
            R.string.option_back_to_home
        )
        popupViewAndWindow.first.findViewById<Button>(R.id.yesButton).setOnClickListener
{
            popupViewAndWindow.second.dismiss()
        }
        popupViewAndWindow.first.findViewById<Button>(R.id.noButton).setOnClickListener
{

findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navigateUp()
            popupViewAndWindow.second.dismiss()
        }
    }

    fun spawnRPDataRemovalPopup(index: Int)
    {
        val rp = _adapter.getItem(index) as RegularPayment
        var popupHeaderId = 0
        var popupTextId = 0
        if (rp.service == RPService.COINBASE.value)
        {
```

Продовження додатка Б

```
            popupHeaderId = R.string.cancel_payment_record_header
            popupTextId = R.string.cancel_payment_record_text
        }
        else
        {
            popupHeaderId = R.string.remove_payment_record_header
            popupTextId = R.string.remove_payment_record_text
        }

        val popupViewAndWindow = spawnPopup(popupHeaderId, popupTextId)
        popupViewAndWindow.first.findViewById<Button>(R.id.yesButton).setOnClickListener
{
            popupViewAndWindow.second.dismiss()
            // todo: toast here instead of return
            val creds = getCurrentUserCreds() ?: return@setOnClickListener
            val item = _adapter.getItem(index) as RegularPayment
            val err =
ServerConnectionHandler.sendRemoveRegularPaymentRequest(creds.first, creds.second,
item.name)
            ServerConnectionHandler.toastIfError(applicationContext, err)
            if (err == ServerConstants.ErrorValue.SUCCESS)
            {
                println("removing: success from server!")
                val db = AppDatabase.getInstance(applicationContext)
                runBlocking {
                    launch(Dispatchers.IO) {
                        db.getRegularPaymentDao().deleteRegularPayment(item)
                    }
                }
                _adapter.removeByIndex(index)
            }
        }
        popupViewAndWindow.first.findViewById<Button>(R.id.noButton).setOnClickListener
{
            popupViewAndWindow.second.dismiss()
        }
    }

    override fun onDestroy()
    {
        endSession()
        super.onDestroy()
    }
}


class LogInActivity : AppCompatActivity()
{
    override fun onCreate(savedInstanceState: Bundle?)
    {
        // REMOVE NEXT 2 LINES ASAP \/
        val policy = StrictMode.ThreadPolicy.Builder().permitAll().build()
        StrictMode.setThreadPolicy(policy)

        setTitle(R.string.log_in_layout_name)

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        findViewById<Button>(R.id.lLogInBtn).setOnClickListener {
```

```
            val username = findViewById<EditText>(R.id.lLoginVal).text.toString()
            val phash = md5(findViewById<EditText>(R.id.lPasswordVal).text.toString())
            val err = ServerConnectionHandler.sendLogInRequest(username, phash)
            ServerConnectionHandler.toastIfError(applicationContext, err)
            if (err == ServerConstants.ErrorValue.SUCCESS)
            {
                println("success!")
                val db = AppDatabase.getInstance(applicationContext)
                runBlocking {
                    launch(Dispatchers.IO) {

db.getSystemSettingDao().insertSetting(SystemSetting(Settings.LAST_USER, username))
                        db.getUserDao().insertUser(User(username, phash))
                        db.getUserDao().getUsers().forEach {
                            println(it.username + "|" + it.passwordHash)
                        }
                    }
                }
                val intent = Intent(this, NavDrawerTestActivity::class.java)
                startActivity(intent)
                finish()
            }
            else
            {
                println("error!")
            }
        }
        findViewById<Button>(R.id.lSignUpBtn).setOnClickListener {
            val intent = Intent(this, SignUpActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}


fun md5(input:String): String
{
    val md = MessageDigest.getInstance("MD5")
    return BigInteger(1, md.digest(input.toByteArray())).toString(16).padStart(32, '0')
}

fun GetCurrentUserCreds(context: Context): Pair<String, String>?
{
    var result: Pair<String, String>? = null
    runBlocking {
        launch(Dispatchers.IO) {
            val db = AppDatabase.getInstance(context)
            val setting = db.getSystemSettingDao().getSetting(Settings.LAST_USER) ?:
return@launch
            val user = db.getUserDao().getUserByUsername(setting.value) ?: return@launch
            result = Pair(user.username, user.passwordHash)
        }
    }
    return result
}


@Entity
```

```
data class User(
    var username: String,
    var passwordHash: String,
    @PrimaryKey(autoGenerate = true) val id: Int? = null
)

object Settings
{
    const val LAST_USER = "last_user"
}

@Entity(indices = [Index(value = ["name"], unique = true)])
data class SystemSetting(
    var name: String,
    var value: String,
    @PrimaryKey(autoGenerate = true) val id: Int? = null
)

enum class RPCurrency(val value: Int)
{
    USD(0),
    UAH(1),
    BTC(2),
    LAST(3);

    companion object
    {
        fun fromInt(value: Int) = values().first { it.value == value }

        fun fromUiStrings(context: Context, string: String): RPCurrency
        {
            val index =
context.resources.getStringArray(R.array.currencies).indexOf(string)
            return if(index == -1) LAST else fromInt(index)
        }

        fun toUiString(context: Context, curr: RPCurrency): String =
context.resources.getStringArray(R.array.currencies)[curr.value]
    }
}

enum class RPRegularity(val value: Int)
{
    MONTHLY(0),
    BIWEEKLY(1),
    WEEKLY(2),
    LAST(3);

    companion object
    {
        fun fromInt(value: Int) = values().first { it.value == value }

        fun fromUiStrings(context: Context, string: String): RPRegularity
        {
            val index =
context.resources.getStringArray(R.array.periods).indexOf(string)
            return if(index == -1) LAST else fromInt(index)
        }
    }
```

```
}

enum class RPService(val value: Int)
{
    NONE(0),
    COINBASE(1),
    LAST(2);

    companion object
    {
        fun fromInt(value: Int) = values().first { it.value == value }

        fun fromUiStrings(context: Context, string: String): RPService
        {
            val index =
context.resources.getStringArray(R.array.payment_systems).indexOf(string)
            return if(index == -1) LAST else fromInt(index + 1) // NONE is not supported
in the UI
        }
    }
}

enum class RPStatus(val value: Int)
{
    SUCCESS(0),
    ERROR(1),
    LAST(2);

    companion object
    {
        fun fromInt(value: Int) = values().first { it.value == value }
    }
}

@Entity
data class RegularPayment(
    var name: String,
    var picUrl: String,
    var currency: Int,
    var amount: Double,
    var regularity: Int,
    var service: Int, // in the past - PaymentSystem
    var status: Int,
    var destination: String? = null,
    @PrimaryKey(autoGenerate = true) val id: Int? = null
) : Serializable


@Database(version = 1, entities = [User::class, SystemSetting::class,
RegularPayment::class], exportSchema = false)
abstract class AppDatabase : RoomDatabase()
{
    abstract fun getUserDao() : UserDao
    abstract fun getSystemSettingDao() : SystemSettingDao
    abstract fun getRegularPaymentDao() : RegularPaymentDao
    companion object
    {
        private var appDatabase: AppDatabase? = null
        fun getInstance(appContext: Context): AppDatabase
```

```
        {
            if (appDatabase == null)
            {
                appDatabase = Room.databaseBuilder(appContext, AppDatabase::class.java,
"testdb8").build()
            }
            return appDatabase!!
        }
    }
}


@Dao
interface UserDao
{
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertUser(user: User)

    @Update
    fun updateUser(user: User)

    @Delete
    fun deleteUser(user: User)

    @Query("select * from user where username = :username limit 1")
    fun getUserByUsername(username: String) : User?

    @Query("select * from user")
    fun getUsers() : Array<User>
}

@Dao
interface SystemSettingDao
{
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertSetting(setting: SystemSetting)

    @Update
    fun updateSetting(setting: SystemSetting)

    @Delete
    fun deleteSetting(setting: SystemSetting)

    @Query("delete from systemsetting where name = :name")
    fun deleteSettingByName(name: String)

    @Query("select * from systemsetting where name = :name limit 1")
    fun getSetting(name: String) : SystemSetting?
}

@Dao
interface RegularPaymentDao
{
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertRegularPayment(rp: RegularPayment)

    @Update
    fun updateRegularPayment(rp: RegularPayment)
```

```
    @Delete
    fun deleteRegularPayment(rp: RegularPayment)

    @Query("select * from regularpayment order by name")
    fun getRegularPayments() : Array<RegularPayment>

    @Query("select * from regularpayment where name = :name limit 1")
    fun getRegularPaymentByName(name: String) : RegularPayment?

    @Query("select * from regularpayment where currency = :currency")
    fun getRegularPaymentsByCurrency(currency: Int) : Array<RegularPayment>

    @Query("delete from regularpayment where 1=1")
    fun deleteAllRegularPayments()
}


class ImportFragment : Fragment() {

    private var _binding: FragmentImportBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    private val parentNavAct get() = activity as NavDrawerTestActivity

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentImportBinding.inflate(inflater, container, false)
        val root: View = binding.root
        binding.saveBtn.setOnClickListener {
            var success = true
            val creds = parentNavAct.getCurrentUserCreds()
            if (creds == null)
                success = false
            else
            {
                val serviceName = binding.serviceNameVal.text.toString()
                val picUrl = binding.imageUrlVal.text.toString()
                val currency = RPCurrency.fromUiStrings(requireContext(),
binding.currencySpin.selectedItem.toString())
                val amount = binding.amountVal.text.toString().toDouble()
                val regularity = RPRegularity.fromUiStrings(requireContext(),
binding.regularitySpin.selectedItem.toString())
                val err =
ServerConnectionHandler.sendAddRegularPaymentRequest(creds.first, creds.second,
serviceName, picUrl, currency, amount, regularity)
                ServerConnectionHandler.toastIfError(requireContext(), err)
                if (err == ServerConstants.ErrorValue.SUCCESS)
                {
                    println("success from server!")
                    val db = AppDatabase.getInstance(requireContext())
                    val rp = RegularPayment(serviceName, picUrl, currency.value, amount,
regularity.value, RPService.NONE.value, RPStatus.SUCCESS.value)
                    runBlocking {
```

Продовження додатка Б

```
                        launch(Dispatchers.IO) {
                            val matchingSetting =
db.getSystemSettingDao().getSetting(Settings.LAST_USER)
                            if (matchingSetting == null)
                            {
                                success = false
                                return@launch
                            }
                            db.getRegularPaymentDao().insertRegularPayment(rp)
                        }
                    }

                    parentNavAct.adapter!!.add(rp)
                }
                else
                    success = false
            }
            println("import result: $success")
            if (success)

parentNavAct.findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navigateU
p()
        }

        return root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}


class ListFragment : Fragment() {

    private var _binding: FragmentListBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    private val parentNavAct get() = activity as NavDrawerTestActivity

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentListBinding.inflate(inflater, container, false)
        val root: View = binding.root

        val listView: RecyclerView = binding.confPartRecyclerView
        listView.layoutManager = LinearLayoutManager(requireContext());
        listView.adapter = parentNavAct.adapter!!;

        regularPaymentModel.getData().observe(requireActivity(), Observer {
            it?.let {
                parentNavAct.adapter!!.updateRegularPayments(it);
```

```
            }
        })
        return root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
    private val regularPaymentModel by lazy {
ViewModelProvider(this).get(RegularPaymentModel::class.java) }
}


class EditFragment(
//    private val regularPayment: RegularPayment
) : Fragment()
{
    private val parentNavAct get() = activity as NavDrawerTestActivity
    private var _binding: FragmentEditBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentEditBinding.inflate(inflater, container, false)
        val root: View = binding.root
        val regularPayment = requireArguments().get("RegularPayment") as RegularPayment
        val index = requireArguments().get("index") as Int
        binding.editServiceNameVal.setText(regularPayment.name)
        binding.editImageUrlVal.setText(regularPayment.picUrl)
        binding.editCurrencySpin.setSelection(regularPayment.currency)
        binding.editAmountVal.setText(regularPayment.amount.toString())
        binding.editRegularitySpin.setSelection(regularPayment.regularity)
        binding.editSaveBtn.setOnClickListener {
            var success = true
            val creds = parentNavAct.getCurrentUserCreds()
            if (creds == null)
                success = false
            else
            {
                val oldName = regularPayment.name
                regularPayment.name = binding.editServiceNameVal.text.toString()
                regularPayment.picUrl = binding.editImageUrlVal.text.toString()
                val currency = RPCurrency.fromUiStrings(requireContext(),
binding.editCurrencySpin.selectedItem.toString())
                regularPayment.currency = currency.value
                regularPayment.amount = binding.editAmountVal.text.toString().toDouble()
                val regularity = RPRegularity.fromUiStrings(requireContext(),
binding.editRegularitySpin.selectedItem.toString())
                regularPayment.regularity = regularity.value
                val removeErr =
ServerConnectionHandler.sendRemoveRegularPaymentRequest(creds.first, creds.second,
oldName)
```

```
                ServerConnectionHandler.toastIfError(requireContext(), removeErr)
                if (removeErr == ServerConstants.ErrorValue.SUCCESS)
                {
                    val addErr =
ServerConnectionHandler.sendAddRegularPaymentRequest(creds.first, creds.second,
regularPayment.name, regularPayment.picUrl, currency, regularPayment.amount, regularity)
                    ServerConnectionHandler.toastIfError(requireContext(), addErr)
                    if (addErr == ServerConstants.ErrorValue.SUCCESS)
                    {
                        println("success from server!")
                        val db = AppDatabase.getInstance(requireContext())
                        runBlocking {
                            launch(Dispatchers.IO) {

db.getRegularPaymentDao().updateRegularPayment(regularPayment)
                            }
                        }
                        parentNavAct.adapter!!.editByIndex(index, regularPayment)
                    }
                    else
                        success = false
                }
                else
                    success = false
            }
            println("edit result: $success")
            if (success)

parentNavAct.findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navigateU
p()
        }

        return root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}


class CreateFragment : Fragment() {

    private var _binding: FragmentCreateBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    private val parentNavAct get() = activity as NavDrawerTestActivity

    fun isAuthorizedOnCoinbase(): Boolean
    {
        val creds = parentNavAct.getCurrentUserCreds()
        if (creds != null)
        {
            val errAndResp =
ServerConnectionHandler.sendCheckIfAuthorizedOnCoinbase(creds.first, creds.second)
```

```
            ServerConnectionHandler.toastIfError(requireContext(), errAndResp.first)
            val resp = errAndResp.second
            if (resp != null && resp.has(ServerConstants.COINBASE_IS_USER_AUTHORIZED) &&
resp.getString(ServerConstants.COINBASE_IS_USER_AUTHORIZED).toBoolean())
            {
                return true
            }
        }
        return false
    }

    @SuppressLint("SimpleDateFormat")
    fun getCurrentDate(): String
    {
        val sdf = SimpleDateFormat("dd/MM/yyyy")
        return sdf.format(Date())
    }

    enum class ButtonVal
    {
        CREATE,
        AUTHORIZE,
        RETRY
    }

    fun turnOnOrOffFields(turnOn: Boolean, btnVal: ButtonVal)
    {
        binding.createServiceNameVal.isEnabled = turnOn
        binding.paymentSystemSpin.isEnabled = turnOn
        binding.createCurrencySpin.isEnabled = turnOn
        binding.createAmountVal.isEnabled = turnOn
        binding.createRegularitySpin.isEnabled = turnOn
        binding.destinationVal.isEnabled = turnOn
        if (btnVal == ButtonVal.CREATE)
            binding.createBtn.text = getString(R.string.create)
        else if (btnVal == ButtonVal.AUTHORIZE)
            binding.createBtn.text = getString(R.string.coinbase_auth)
        else if (btnVal == ButtonVal.RETRY)
            binding.createBtn.text = getString(R.string.retry_payment)
    }

    fun set2faFieldisibility(visible: Boolean)
    {
        val visibility = if (visible) View.VISIBLE else View.INVISIBLE
        binding.tfaCodeVal.visibility = visibility
        //binding.createBtn.text = if(visible) getString(R.string.create) else
getString(R.string.coinbase_auth)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCreateBinding.inflate(inflater, container, false)
        val root: View = binding.root

        set2faFieldisibility(false)
        if (!isAuthorizedOnCoinbase())
```

```
        {
            turnOnOrOffFields(false, ButtonVal.AUTHORIZE)
        }

        val existingRP = arguments?.get("RegularPayment") as RegularPayment?
        val index = arguments?.get("index") as Int?
        if (existingRP != null)
        {
            binding.createServiceNameVal.setText(existingRP.name)
            binding.createRegularitySpin.setSelection(existingRP.regularity)

            // todo: temporarily index is 0
            binding.createCurrencySpin.setSelection(0)
            binding.paymentSystemSpin.setSelection(0)

            binding.createAmountVal.setText(existingRP.amount.toString())
            binding.destinationVal.setText(existingRP.destination)
            turnOnOrOffFields(false, ButtonVal.RETRY)
        }

        binding.createBtn.setOnClickListener {
            if (isAuthorizedOnCoinbase())
            {
                val nonePicUrl = "None"
                val creds = parentNavAct.getCurrentUserCreds()!!

                val serviceName = binding.createServiceNameVal.text.toString()
                val currency = RPCurrency.fromUiStrings(requireContext(),
binding.createCurrencySpin.selectedItem.toString())
                val amount = binding.createAmountVal.text.toString().toDouble()
                val regularity = RPRegularity.fromUiStrings(requireContext(),
binding.createRegularitySpin.selectedItem.toString())
                val destination = binding.destinationVal.text.toString()
                val service = RPService.fromUiStrings(requireContext(),
binding.paymentSystemSpin.selectedItem.toString())

                var tfa: String? = null
                if (binding.tfaCodeVal.visibility == View.VISIBLE &&
binding.tfaCodeVal.text.toString().isNotEmpty())
                    tfa = binding.tfaCodeVal.text.toString()


                val resultPair =
ServerConnectionHandler.sendCreateRegularPaymentRequest(creds.first, creds.second,
serviceName, nonePicUrl, currency, amount, regularity, destination, getCurrentDate(),
existingRP != null, tfa)
                ServerConnectionHandler.toastIfError(requireContext(), resultPair.first,
resultPair.second)
                if (resultPair.first == ServerConstants.ErrorValue.RESEND_WITH_TOKEN)
                {
                    set2faFieldisibility(true)
                    return@setOnClickListener
                }
                val rp = existingRP ?: RegularPayment(serviceName, nonePicUrl,
currency.value, amount, regularity.value, service.value, RPStatus.SUCCESS.value,
destination)
                if (resultPair.first == ServerConstants.ErrorValue.SUCCESS)
                {
                    println("success from server!")
```

```
                }
                else
                {
                    rp.status = RPStatus.ERROR.value
//                    Toast.makeText(requireContext(), "Transaction failed: " +
resultPair.second, Toast.LENGTH_LONG).show()
                }

                // insert anyway, but with correct status
                val db = AppDatabase.getInstance(requireContext())
                var added = false
                runBlocking {
                    launch(Dispatchers.IO) {
                        db.getSystemSettingDao().getSetting(Settings.LAST_USER) ?:
return@launch
                        db.getRegularPaymentDao().insertRegularPayment(rp)
                        added = true
                    }
                }
                if (added)
                {
                    if (existingRP == null) // add operation
                        parentNavAct.adapter!!.add(rp)
                    else // edit operation
                        parentNavAct.adapter!!.editByIndex(index!!, rp)
                }
            }
            else
            {
                val creds = parentNavAct.getCurrentUserCreds()
                if (creds != null)
                {
                    val req =
ServerConnectionHandler.composeCoinbaseAuthRequest(creds.first, creds.second)
                    val browserIntent = Intent(Intent.ACTION_VIEW, Uri.parse(req))
                    startActivity(browserIntent)
                }
            }

parentNavAct.findNavController(R.id.nav_host_fragment_content_nav_drawer_test).navigateU
p()
        }

        return root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}


class CalcSpendingsFragment : Fragment()
{
    private var _binding: FragmentCalcSpendingsBinding? = null

    private val binding get() = _binding!!
```

Закінчення додатка Б

```
    private val parentNavAct get() = activity as NavDrawerTestActivity

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCalcSpendingsBinding.inflate(inflater, container, false)
        val root: View = binding.root

        binding.calcBtn.setOnClickListener {
            var found: Array<RegularPayment>? = null
            runBlocking {
                launch(Dispatchers.IO) {
                    found =
AppDatabase.getInstance(requireContext()).getRegularPaymentDao().getRegularPaymentsByCur
rency(RPCurrency.fromUiStrings(requireContext(),
binding.calcCurrencySpin.selectedItem.toString()).value)
                }
            }
            var sum: Double = 0.0
            var result: Double = 0.0
            found!!.forEach {
                var multiplier = 0
                when (it.regularity) {
                    RPRegularity.MONTHLY.value -> multiplier = 1
                    RPRegularity.BIWEEKLY.value -> multiplier = 2
                    RPRegularity.WEEKLY.value -> multiplier = 4
                }
                sum += it.amount * multiplier
            }
            println("sum: $sum")
            when (RPRegularity.fromUiStrings(requireContext(),
binding.calcRegularitySpin.selectedItem.toString())) {
                RPRegularity.MONTHLY -> result = sum
                RPRegularity.BIWEEKLY -> result = sum / 2
                RPRegularity.WEEKLY -> result = sum / 4
            }
            parentNavAct.spawnCalcSpendingPopup(result,
binding.calcCurrencySpin.selectedItem.toString())
        }

        return root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```