МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

ОМЕЛЯНЧУК ВОЛОДИМИР ВОЛОДИМИРОВИЧ

РОЗРОБКА МУЛЬТИКОРИСТУВАЦЬКОЇ МОБІЛЬНОЇ СИСТЕМИ БЮДЖЕТНОГО КОНТРОЛЮ З ФУНКЦІЯМИ ЗВІТНОСТІ ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ SPRING

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
Булатецький Віталій Вікторович,
кандидат фізико-математичних наук,
доцент кафедри комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол № _____
Засідання кафедри комп’ютерних наук
та кібербезпеки
Від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК 2025

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 АНАЛІЗ ТА ОСОБЛИВОСТІ БЮДЖЕТНИХ СИСТЕМ 5
1.1. Розвиток цифрових фінансових технологій 5
1.2. Особливості багатокористувацьких фінансових систем 7
1.2.1. Управління спільними бюджетами 7
1.2.2. Взаємодія користувачів у фінансових додатках 10
1.2.3. Безпека фінансових транзакцій 11
1.3. Архітектурні підходи до створення мобільних бюджетних систем 13
1.3.1. Мікросервісна архітектура у фінансових додатках 13
1.3.2. Використання Spring Boot для серверної частини 14
1.3.3. Робота з реляційними базами даних на прикладі PostgreSQL 16
1.3.4. Забезпечення безпеки за допомогою Spring Security та JWT 18
1.4. Огляд існуючих мобільних систем бюджетного контролю та їхніх функціональних можливостей 20
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОЇ СИСТЕМИ БЮДЖЕТНОГО КОНТРОЛЮ З ФУНКЦІЯМИ ЗВІТНОСТІ ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ SPRING 24
2.1. Постановка задачі, призначення та вимоги до програмного засобу 24
2.2. Вибір моделі розробки програмного засобу 25
2.3. Архітектура та структура застосунку 26
2.3.1. Серверна частина на базі Spring Boot 27
2.3.2. Клієнтська частина: мобільний додаток на Kotlin 29
2.3.3. Архітектура та реалізація бази даних на PostgreSQL 31
2.4. Обґрунтування вибору технологій та інструментів для системи бюджетного контролю 33
2.5. Особливості програмної реалізації системи бюджетного контролю 35
2.6. Організація тестування та налагодження програмного засобу 38
2.7. Рекомендації щодо використання та впровадження мобільного додатка 39

ВИСНОВКИ..... 43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 45
ДОДАТКИ..... 47

ВСТУП

Актуальність теми. Сучасні умови економічної діяльності висувають до суб’єктів господарювання дедалі жорсткіші вимоги щодо ефективного контролю над фінансовими потоками та своєчасної аналітики бюджетних показників. Мобільні рішення, сполучені зі стабільною серверною інфраструктурою, відкривають нові можливості для оперативного моніторингу витрат і доходів у режимі реального часу.

Метою роботи є розробка мультикористувацької мобільної системи бюджетного контролю з функціоналом генерації та представлення звітності, що забезпечить зручний інтерфейс для кінцевого користувача і стійку бекенд-логіку на основі Spring.

Завдання роботи. У межах дослідження передбачено формулювання вимог до користувацького інтерфейсу й серверної архітектури, побудову моделі даних для транзакцій та категорій витрат і доходів, реалізацію механізмів аутентифікації й авторизації, інтеграцію модулів генерації звітів у різних форматах та перевірку працездатності системи під навантаженням.

Об’єктом дослідження виступає інформаційна система, що забезпечує управління бюджетом користувачів із різними рівнями доступу, включно з механізмами віддаленого зберігання й обробки фінансових даних.

Предметом дослідження є методи та технології побудови мультикористувацьких мобільних застосунків із використанням Spring Boot на серверній частині та Android-клієнта, а також підходи до формування і відображення звітної інформації в мобільному середовищі.

РОЗДІЛ 1
АНАЛІЗ ТА ОСОБЛИВОСТІ БЮДЖЕТНИХ СИСТЕМ

1.1. Розвиток цифрових фінансових технологій

Розвиток цифрових фінансових технологій є важливим чинником, що сприяє трансформації сучасного фінансового сектора. За останні десятиліття цей процес змінив традиційні підходи до обслуговування клієнтів, оптимізував внутрішні бізнес-процеси та відкрив нові можливості для інтеграції фінансових послуг у повсякденне життя. Перші кроки у цифровізації фінансів були спрямовані на автоматизацію банківських операцій і впровадження електронних платіжних систем, що дозволило спростити виконання транзакцій і знизити операційні витрати. Розвиток інтернет-технологій сприяв появі онлайн-банкінгу, який зробив доступ до фінансових послуг більш зручним та оперативним, дозволивши користувачам здійснювати операції незалежно від місця перебування.

Подальша еволюція технологій відзначалася впровадженням мобільних додатків, що забезпечують постійний зв’язок користувачів із фінансовими установами. Завдяки розповсюдженню смартфонів, мобільний банкінг набув широкої популярності, що значно спростило процес управління фінансами для мільйонів людей по всьому світу. Сучасні рішення дозволяють не тільки проводити операції, але й аналізувати витрати, планувати бюджет та отримувати детальну звітність. Такий підхід сприяє більшій прозорості та контролю над фінансовими потоками як для фізичних осіб, так і для корпоративних клієнтів.

Не менш важливим аспектом у розвитку цифрових фінансових технологій стало питання безпеки. Сучасні системи використовують різноманітні методи захисту даних, серед яких криптографія, багатофакторна аутентифікація та інноваційні протоколи шифрування, що дозволяють запобігати несанкціонованому доступу до конфіденційної інформації. Постійне вдосконалення заходів кібербезпеки стало необхідною умовою для забезпечення

стабільної роботи фінансових систем, адже зростання кількості цифрових транзакцій супроводжується зростанням кіберзагроз. Розробка ефективних механізмів захисту сприяє зростанню довіри користувачів до електронних сервісів та стимулює подальший розвиток цифрових технологій у фінансовій сфері [18].

Сучасні цифрові фінансові технології також відзначаються високою гнучкістю та масштабованістю завдяки застосуванню хмарних обчислень і мікросервісних архітектур. Це дозволяє системам працювати в режимі реального часу, обробляти великі обсяги даних та швидко адаптуватися до змін ринкових умов. Такі технології відкривають нові можливості для створення інтегрованих платформ, які об’єднують різні фінансові інструменти в єдину систему, забезпечуючи зручний доступ до інформації та управління бюджетом. Розробка мобільних додатків з функціями звітності стає логічним продовженням цієї тенденції, що дозволяє не лише контролювати поточні витрати, але й прогнозувати майбутні фінансові потреби.

Сучасний етап розвитку цифрових фінансових технологій характеризується інтеграцією інноваційних рішень, спрямованих на підвищення якості фінансових послуг. Важливу роль у цьому процесі відіграють стартапи та технологічні компанії, які активно впроваджують нові підходи до взаємодії з користувачами. Ринок фінансових технологій стимулює конкуренцію між традиційними банками та новими гравцями, що сприяє швидкому впровадженню інновацій і покращенню сервісів. Завдяки цьому, цифровізація фінансів стає не лише технічним досягненням, але й фактором економічного зростання та розвитку нових бізнес-моделей.

Підсумовуючи, розвиток цифрових фінансових технологій охоплює широке коло аспектів, що включають модернізацію банківських послуг, впровадження мобільних технологій, підвищення рівня безпеки та оптимізацію операцій за допомогою сучасних ІТ-рішень. Цей процес відкриває нові горизонти для управління фінансовими ресурсами, робить фінансові сервіси

доступнішими і надійнішими, що є надзвичайно важливим у сучасному швидкозмінному світі [19].

1.2. Особливості багатокористувацьких фінансових систем

1.2.1. Управління спільними бюджетами

У сучасних цифрових умовах управління спільними бюджетами в багатокористувацьких фінансових системах набуває дедалі більшої актуальності. Це зумовлено як зростаючою популярністю колективного підходу до фінансового планування, так і необхідністю підвищення прозорості та ефективності використання фінансових ресурсів. У міру того як сім'ї, друзі, бізнес-команди та інші спільноти стикаються з необхідністю спільного розподілу витрат, потреба в системах, що дозволяють ефективно координувати фінансові потоки, автоматизувати облік і гарантувати прозорість, стає особливо важливою.

Основна мета таких систем полягає в оптимізації використання наявних коштів, забезпеченні чіткого контролю за доходами та витратами, а також у формуванні довіри між усіма учасниками. Сучасні фінансові застосунки підтримують розширений функціонал, що включає автоматичне фіксування транзакцій, категоризацію витрат, формування аналітичних звітів, а також зручні візуалізації даних для кращого розуміння фінансового стану групи. Однією з важливих переваг є можливість інтеграції з банківськими сервісами, платіжними платформами та іншими зовнішніми джерелами даних, що дозволяє системам оновлювати інформацію в режимі реального часу та забезпечувати її точність і актуальність.

У таких багатокористувацьких середовищах важливу роль відіграє структура доступу до функціоналу системи. Розподіл ролей серед користувачів є одним із ключових аспектів організації фінансової взаємодії. Як правило, система передбачає наявність адміністратора або групи адміністраторів, які мають розширені права управління — включаючи створення бюджетів, затвердження транзакцій, керування доступом інших користувачів та

налаштування параметрів групи. Решта учасників зазвичай отримують обмежений доступ відповідно до призначених їм ролей — наприклад, можливість додавати власні витрати, переглядати загальну статистику або коментувати певні фінансові дії.

Такий розподіл повноважень сприяє впорядкованості процесу взаємодії у групі, дозволяючи уникнути ситуацій, коли окремі учасники можуть випадково або навмисно змінити важливі фінансові параметри. Крім того, завдяки чіткому контролю доступу знижується ризик помилкових дій, що можуть призвести до фінансових втрат або конфліктів між користувачами. Застосування принципу "найменш необхідних прав" дозволяє обмежити дії кожного користувача лише тими, які необхідні для виконання його ролі, тим самим підвищуючи загальний рівень безпеки.

Інший важливий аспект – це система сповіщень і повідомлень. Інтегровані механізми повідомлень дозволяють учасникам миттєво дізнаватися про зміни у спільному бюджеті, нові транзакції, запити на підтвердження витрат або наближення до встановлених лімітів. Це сприяє оперативному прийняттю рішень, зменшує ризик непорозумінь та затримок у реагуванні. Особливо важливо це у великих групах, де кожне фінансове рішення може мати значний вплив на загальний баланс і стосунки між учасниками.

Безпека інформації в багатокористувацьких системах управління бюджетом має вирішальне значення. Адже в системах такого типу обробляється чутлива фінансова інформація, включаючи особисті дані, банківські реквізити та історію транзакцій. Для гарантування захисту даних впроваджуються сучасні методи шифрування як під час збереження, так і при передаванні інформації. Багатофакторна аутентифікація (2FA) використовується для додаткового підтвердження особи користувача при вході в систему або здійсненні важливих фінансових дій. Це дозволяє зменшити ймовірність несанкціонованого доступу навіть у разі компрометації пароля.

Окрему увагу варто приділити механізмам аудиту, які дають змогу відстежити всі дії, що були здійснені в системі. Журнали активності фіксують

кожну транзакцію, зміну ролей, налаштувань або доступу, що значно полегшує розслідування у разі виникнення суперечок або технічних збоїв. Можливість зберігати історію змін дозволяє не лише підвищити прозорість, але й слугує основою для подальшого аналізу та вдосконалення системи управління фінансами в групі.

У сучасному середовищі мобільних технологій зручність використання є важливою умовою успіху будь-якої фінансової системи. Саме тому інтерфейси мобільних застосунків розробляються з урахуванням принципів UX/UI дизайну, щоб забезпечити інтуїтивну навігацію, швидкий доступ до основного функціоналу та адаптацію під різні сценарії використання. Підтримка кросплатформенності дозволяє користувачам взаємодіяти з системою як через смартфони, так і через веб-інтерфейс, що забезпечує гнучкість і доступність у будь-який час і в будь-якому місці.

Ще однією перевагою сучасних фінансових систем є наявність аналітичних інструментів, які дозволяють не лише контролювати витрати, але й формувати прогнози, встановлювати фінансові цілі, порівнювати бюджети між місяцями або групами, а також виявляти закономірності у фінансовій поведінці. Такі можливості роблять багатокористувацькі системи не просто засобом для фіксації транзакцій, а повноцінним інструментом для прийняття обґрунтованих рішень та стратегічного планування.

Варто зазначити, що впровадження подібних систем значно покращує комунікацію між учасниками. Завдяки чітко структурованим звітам, можливості залишати коментарі до транзакцій, налаштуванням нагадувань і погоджень, користувачі можуть обговорювати та координувати спільні фінансові дії у зручному та прозорому форматі. Це особливо корисно у групах, де учасники фізично віддалені один від одного або мають різний рівень фінансової грамотності.

У підсумку, багатокористувацькі системи управління спільними бюджетами поєднують у собі технологічні досягнення, безпекові стандарти та потреби користувачів у зручності, гнучкості та прозорості. Вони забезпечують

не лише ефективне використання фінансових ресурсів, але й створюють простір для співпраці, планування та розвитку фінансової культури серед широкого кола користувачів. Усе це робить такі системи важливою складовою цифрової трансформації у сфері персональних і групових фінансів, зокрема в умовах швидких змін ринку та зростання значення фінансової незалежності [20].

1.2.2. Взаємодія користувачів у фінансових додатках

Взаємодія користувачів у фінансових додатках є важливою складовою успішної роботи будь-якого рішення, спрямованого на управління коштами. Сучасні додатки забезпечують інтуїтивно зрозумілий інтерфейс, який дозволяє легко орієнтуватися у функціоналі та оперативно виконувати необхідні операції. Оптимізація взаємодії досягається за рахунок мінімізації кількості кроків для виконання транзакцій, що сприяє зменшенню часу обробки даних та підвищенню користувацького комфорту. Важливим елементом є адаптивний дизайн, який забезпечує зручність роботи як на мобільних пристроях, так і на комп’ютерах, враховуючи специфіку кожної платформи.

Системи фінансових додатків впроваджують механізми миттєвих сповіщень, що дозволяють користувачам своєчасно отримувати інформацію про проведені операції, зміни балансу або заплановані платежі. Такий підхід забезпечує оперативне реагування на фінансові події, мінімізуючи затримки в обміні даними між користувачами та серверами додатка. Крім того, сучасні технології дозволяють налаштовувати параметри сповіщень відповідно до індивідуальних потреб, що сприяє більш точному інформуванню та підвищенню ефективності використання ресурсу.

Взаємодія між користувачами набуває особливої важливості в умовах спільного управління фінансами, коли група осіб бере участь у прийнятті рішень щодо розподілу коштів. Додатки, орієнтовані на спільні бюджетні операції, надають можливість вести обговорення, залишати коментарі та проводити голосування, що дозволяє формувати злагоджену стратегію розподілу ресурсів. Такий механізм сприяє прозорості фінансових операцій та забезпечує довіру між

учасниками, адже кожна транзакція підтверджується спільно та ретельно фіксується в системі.

Інтеграція аналітичних інструментів дозволяє користувачам здійснювати моніторинг фінансових показників у реальному часі, що сприяє прийняттю обґрунтованих рішень. Аналіз даних з історії операцій допомагає виявляти закономірності, оптимізувати витрати та планувати подальші фінансові кроки. Використання таких інструментів є важливим чинником у підвищенні якості взаємодії, оскільки кожен користувач отримує можливість адаптувати функціональність додатку під свої потреби.

Сучасні фінансові додатки демонструють тенденцію до персоналізації, завдяки чому кожен користувач може налаштовувати свій робочий простір відповідно до індивідуальних вимог. Персоналізований інтерфейс не лише полегшує доступ до важливих функцій, але й забезпечує високий рівень зручності в управлінні фінансовими потоками. В результаті, взаємодія між користувачами стає більш продуктивною, що позитивно впливає на загальну ефективність використання додатку [5].

1.2.3. Безпека фінансових транзакцій

Безпека фінансових транзакцій є основним елементом довіри до цифрових фінансових систем. Захист даних, що передаються між користувачами та сервером, забезпечує цілісність та конфіденційність операцій, що має вирішальне значення для збереження фінансової стабільності. Розробка надійних механізмів безпеки передбачає використання сучасних криптографічних технологій, які дозволяють шифрувати дані як під час передачі, так і при зберіганні. Захищений канал комунікації, реалізований за допомогою протоколів SSL/TLS, мінімізує ризик перехоплення інформації, а застосування асиметричного та симетричного шифрування сприяє оптимальному поєднанню продуктивності та надійності захисту [6].

Сучасні системи забезпечення безпеки впроваджують багатофакторну аутентифікацію, що значно ускладнює доступ до конфіденційних даних з боку

зловмисників. Поєднання паролів, одноразових кодів та біометричних даних дозволяє знизити ймовірність несанкціонованого входу навіть у випадку компрометації одного з факторів. Використання токенів, зокрема JWT, дозволяє керувати сесіями користувачів, забезпечуючи безперервний контроль над тим, хто має доступ до системи, та швидко реагувати на спроби несанкціонованого доступу.

Не менш важливим аспектом є системи моніторингу та аудиту транзакцій, які дозволяють виявляти підозрілу активність в режимі реального часу. Автоматизовані алгоритми аналізу поведінки користувачів сприяють виявленню аномалій, що можуть свідчити про шахрайські дії або кібератаки. Регулярне сканування мережевих з'єднань, аналіз логів операцій допомагають своєчасно виявляти потенційні загрози та усувати їх до того, як вони завдадуть шкоди системі [14].

Впровадження принципів DevSecOps у процес розробки сприяє інтеграції заходів безпеки на кожному етапі життєвого циклу програмного продукту. Це дозволяє не лише оперативно виявляти вразливості у програмному коді, але й своєчасно впроваджувати оновлення. Такий підхід забезпечує безперервне вдосконалення захисних механізмів та зменшує можливість використання знайдених недоліків у системі.

Регулярне навчання персоналу та користувачів основам кібербезпеки є додатковим заходом, що знижує ризик помилок, спричинених людським фактором. Інформування про сучасні методи атак і способи їх запобігання сприяє формуванню обізнаності, що в умовах зростання кількості транзакцій стає невід'ємною складовою загального захисту системи.

Забезпечення безпеки фінансових транзакцій охоплює комплекс заходів, які включають шифрування даних, багатофакторну аутентифікацію, системи моніторингу та інтеграцію безпеки у всі етапи розробки. Цей підхід дозволяє створити надійну інфраструктуру, що захищає користувачів від кібератак та сприяє стабільній роботі цифрових фінансових систем. Надійний захист

транзакцій є запорукою довіри користувачів, що стимулює подальший розвиток технологій та впровадження інноваційних рішень у сфері фінансів [15].

1.3. Архітектурні підходи до створення мобільних бюджетних систем

1.3.1. Мікросервісна архітектура у фінансових додатках

Мікросервісна архітектура у фінансових додатках стає основою для створення гнучких, масштабованих і стійких систем, що відповідають високим вимогам сучасного ринку. Цей підхід передбачає розбиття великої системи на незалежні, спеціалізовані сервіси, кожен з яких відповідає за окрему бізнес-функцію. Така структура дозволяє ізолювати помилки, проводити оновлення без зупинки всієї системи та ефективно управляти ресурсами при зростанні навантаження.

Фінансові додатки зазвичай працюють з великим обсягом транзакцій, що вимагає високої доступності та безперервності роботи. Мікросервісна архітектура забезпечує можливість розподілу обчислювального навантаження, коли кожен сервіс може масштабуватися окремо. Наприклад, у системі управління бюджетом окремі сервіси можуть відповідати за обробку платежів, автентифікацію користувачів, генерацію звітів або інтеграцію з зовнішніми банківськими сервісами. Така децентралізація дозволяє не лише оптимізувати продуктивність, а й підвищити стійкість системи: збій одного мікросервісу не призводить до повного припинення роботи всього додатку.

Важливу роль у впровадженні мікросервісної архітектури відіграють сучасні інструменти та платформи розробки, зокрема Spring Boot та Spring Cloud. Spring Boot сприяє швидкому створенню автономних сервісів з мінімальними налаштуваннями, а Spring Cloud надає готові рішення для конфігурації, маршрутизації запитів і управління сервісами у розподіленому середовищі. Використання цих технологій дозволяє розробникам зосередитися на реалізації бізнес-логіки, водночас забезпечуючи інтеграцію та стійкість системи [1].

Крім того, сучасні фінансові додатки використовують контейнери, такі як Docker, разом із системами оркестрації, наприклад Kubernetes, для автоматизованого розгортання та масштабування мікросервісів. Цей підхід дозволяє оперативно реагувати на зміну навантаження, забезпечуючи безперебійну роботу при пікових навантаженнях. Сервіси можуть автоматично перезапускатися у разі збоїв, що значно підвищує загальну надійність системи.

Прикладом успішного впровадження мікросервісної архітектури може служити система обробки платіжних операцій, де окремі сервіси відповідають за валідацію транзакцій, управління ризиками, облік платежів та взаємодію з банківськими API. Така децентралізована структура дозволяє забезпечити високий рівень безпеки та оперативно виявляти потенційні загрози за допомогою спеціалізованих сервісів моніторингу та аудиту.

Однак, перехід до мікросервісної архітектури вимагає ретельного планування інтеграції компонентів. Забезпечення ефективної взаємодії між мікросервісами потребує використання стандартизованих протоколів, таких як REST API чи gRPC, а також реалізації механізмів централізованого логування та моніторингу. Ці заходи дозволяють відстежувати роботу кожного сервісу, швидко виявляти помилки та оптимізувати комунікаційні процеси.

Таким чином, мікросервісна архітектура у фінансових додатках є потужним інструментом для підвищення гнучкості, масштабованості та надійності систем. Вона дозволяє ефективно розподіляти обчислювальні ресурси, ізолювати помилки та швидко впроваджувати інноваційні рішення, що є надзвичайно важливим у сучасному фінансовому середовищі [11].

1.3.2. Використання Spring Boot для серверної частини

Spring Boot значно спрощує розробку серверної частини завдяки своїй автоматичній конфігурації та вбудованим засобам для швидкого розгортання додатків. Фреймворк забезпечує вбудований веб-сервер, зазвичай Tomcat або Jetty, що дозволяє розробникам негайно запустити проект без необхідності окремої інсталяції серверного середовища. Це особливо корисно при створенні

фінансових додатків, де швидкість розгортання та адаптивність системи мають вирішальне значення.

Spring Boot інтегрується з різноманітними модулями екосистеми Spring, що дозволяє ефективно управляти залежностями, обробкою REST-запитів, безпекою та роботою з базами даних. Використання Spring Data сприяє простій інтеграції як реляційних, так і NoSQL баз даних, забезпечуючи зручну реалізацію операцій CRUD. Завдяки підтримці JPA та транзакційності, додаток забезпечує високу узгодженість даних, що є критично важливим у фінансових системах [3].

Однією з помітних переваг є можливість створення окремих модулів для різних бізнес-функцій. Наприклад, окремий сервіс може відповідати за аутентифікацію та авторизацію користувачів, інший – за обробку платежів, а третій – за генерацію звітності. Така модульна структура дозволяє ізолювати окремі компоненти, полегшуючи їх масштабування та підтримку без впливу на всю систему.

Spring Boot активно підтримує інтеграцію з Spring Security, що дозволяє впровадити сучасні механізми захисту даних. Багатофакторна аутентифікація, використання токенів для керування сесіями та реалізація політик доступу допомагають забезпечити надійний захист інформації, що обробляється у фінансових транзакціях. Крім того, Spring Boot легко інтегрується з Spring Cloud, що дозволяє розробляти мікросервісні архітектури та впроваджувати автоматизовані стратегії розгортання у хмарному середовищі [4].

Контейнеризація додатків за допомогою Docker у поєднанні зі Spring Boot дозволяє створювати ізольовані середовища для тестування та розгортання, що сприяє швидкому впровадженню оновлень та масштабуванню системи. Це особливо актуально для фінансових сервісів, де стабільність роботи та можливість оперативно реагувати на зміну навантаження мають першорядне значення.

Отже, використання Spring Boot для серверної частини фінансових додатків сприяє створенню надійних, масштабованих та безпечних систем. Завдяки автоматичній конфігурації, модульній архітектурі та інтеграції з іншими

компонентами екосистеми Spring, розробники можуть швидко впроваджувати інноваційні рішення, що відповідають високим вимогам сучасного ринку фінансових технологій [16].

1.3.3. Робота з реляційними базами даних на прикладі PostgreSQL

Робота з реляційними базами даних є однією з основних складових розробки фінансових додатків, оскільки забезпечує надійне зберігання, обробку та аналіз критично важливої інформації. PostgreSQL, як потужна система управління базами даних з відкритим кодом, завдяки своїй високій продуктивності, стабільності та відповідності принципам ACID, стає вибором номер один для багатьох проектів у фінансовій сфері.

У сучасних фінансових додатках PostgreSQL використовується для збереження транзакційних даних, інформації про клієнтів, операцій та звітів. Завдяки підтримці складних SQL-запитів, розширеним можливостям індексації та оптимізації запитів, ця СУБД дозволяє ефективно працювати з великими обсягами даних, що є надзвичайно важливим у системах, де кожна секунда може визначати успіх операції. Особливу увагу приділено забезпеченню узгодженості даних і високій доступності, що досягається за рахунок реалізації кластеризації та реплікації.

Підключення до PostgreSQL у Spring Boot здійснюється через стандартну конфігурацію, що забезпечує гнучкість і простоту налаштування додатку. Наприклад, для встановлення з’єднання з базою даних можна використати наступну конфігурацію у файлі application.properties:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/finance_db
spring.datasource.username=finance_user
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

У цьому прикладі вказано URL для з’єднання з базою даних finance_db, яка розташована на локальному сервері, а також облікові дані користувача finance_user. Встановлення параметру драйвера гарантує правильне завантаження необхідних бібліотек, а налаштування Hibernate дозволяє автоматично формувати схему бази даних відповідно до сутностей, що значно спрощує процес розробки та тестування [9].

Використання Spring Data JPA разом з PostgreSQL забезпечує розробникам зручний доступ до методів CRUD через розширення інтерфейсів типу JpaRepository. Це дозволяє зосередитися на реалізації бізнес-логіки, не витрачаючи час на написання шаблонного коду для взаємодії з базою даних. Крім того, можливість застосування кастомних запитів, а також підтримка транзакційності дозволяють реалізувати складні бізнес-процеси з високою точністю та надійністю.

Однією з важливих переваг PostgreSQL є його здатність працювати з великими обсягами даних та забезпечувати високу продуктивність завдяки оптимізованим алгоритмам збереження і пошуку інформації. Це особливо актуально для фінансових систем, де час реакції та точність обчислень є критично важливими. Інтеграція PostgreSQL з Spring Boot дозволяє ефективно масштабувати систему, використовуючи можливості горизонтального та вертикального масштабування, що забезпечує безперебійну роботу додатку навіть під час пікових навантажень.

Таким чином, використання PostgreSQL у фінансових додатках на базі Spring Boot створює надійну, масштабовану та гнучку систему для управління фінансовими даними. Наявність простих засобів налаштування через application.properties, потужна підтримка транзакцій та можливість роботи з великими обсягами інформації сприяють ефективному впровадженню інноваційних рішень і підвищенню загальної якості фінансових сервісів [10].

1.3.4. Забезпечення безпеки за допомогою Spring Security та JWT

Безпека фінансових додатків є критично важливим аспектом, оскільки вони працюють з конфіденційною інформацією користувачів і фінансовими транзакціями. Для захисту серверної частини таких систем у Spring Boot широко використовується Spring Security у поєднанні з JWT (JSON Web Token). Цей підхід забезпечує надійний механізм автентифікації та авторизації, що дозволяє ефективно захищати API-запити та запобігати несанкціонованому доступу.

Spring Security надає потужний набір інструментів для контролю доступу до ресурсів додатка. Використання JWT дозволяє реалізувати безсерверну автентифікацію, де замість збереження стану сесій на сервері система використовує цифрово підписані токени. Це забезпечує гнучкість і масштабованість, оскільки клієнтський додаток може надсилати токен у кожному запиті, уникаючи необхідності зберігати інформацію про сесії на сервері [13].

Процес автентифікації та авторизації за допомогою Spring Security та JWT виглядає так:

а) користувач вводить свої облікові дані, які надсилаються на сервер у запиті до ендпоінту /login;

б) сервер перевіряє дані за допомогою UserDetailsService та, якщо вони коректні, генерує JWT-токен, який підписується секретним ключем;

в) клієнт отримує токен і надсилає його у заголовку Authorization: Bearer <token> у всіх наступних запитах;

г) Spring Security перехоплює запити, перевіряє валідність токена та визначає права доступу користувача [8].

Приклад конфігурації безпеки у Spring Security з JWT у класі SecurityConfig:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/login", "/register").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public JWTAuthenticationFilter jwtAuthenticationFilter() {
        return new JWTAuthenticationFilter();
    }
}
```

Генерація JWT-токена здійснюється за допомогою бібліотеки io.jsonwebtoken.Jwts. Наприклад:

```
public String generateToken(String username) {
    return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 86400000)) // 1 день
            .signWith(SignatureAlgorithm.HS256, "secretKey")
            .compact();
```

```
}
```

JWT не тільки захищає додаток, але й спрощує управління правами доступу. Наприклад, у фінансовій системі можна створити ролі користувачів (USER, ADMIN), і контролювати доступ до певних ресурсів за допомогою анотацій:

```
@PreAuthorize("hasRole('ADMIN')")
public void generateFinancialReport() {
   // Код генерації звіту }
```

Використання Spring Security у поєднанні з JWT дозволяє створити надійний механізм захисту, який підходить для багатокористувацьких фінансових систем. Безсерверний підхід покращує продуктивність, а цифрово підписані токени гарантують цілісність та безпеку даних, що робить цю комбінацію ефективним рішенням для захисту бюджетних мобільних додатків [6].

1.4. Огляд існуючих мобільних систем бюджетного контролю та їхніх функціональних можливостей

Бюджетні трекери стали невід’ємною частиною фінансового планування, дозволяючи користувачам ефективно контролювати витрати, доходи та накопичення. Сучасні фінансові додатки забезпечують автоматизацію обліку коштів, зручну категоризацію транзакцій, створення звітів та інтеграцію з банківськими сервісами. Серед великої кількості доступних рішень можна виокремити такі популярні додатки, як YNAB (You Need a Budget) та Splitwise, які пропонують різні підходи до управління фінансами, орієнтуючись на різні потреби користувачів [2].

YNAB (You Need A Budget) – розроблений компанією You Need A Budget LLC, цей мобільний та веб-додаток використовує принцип бюджетування з нульовим балансом, де кожен долар має своє призначення. Реалізований мовами JavaScript та Python, YNAB підтримує синхронізацію між пристроями та пропонує навчальні матеріали для користувачів. Додаток орієнтований на покращення фінансових звичок, надає звіти про витрати й дозволяє планувати

витрати з урахуванням реальних потреб. Завдяки регулярному оновленню та підтримці спільноти, YNAB став одним із найпопулярніших інструментів для персонального бюджетування. Основні функції:

1) відстеження доходів і витрат;

2) автоматична категоризація транзакцій;

3) планування бюджету за нульовим принципом;

4) інтеграція з банками;

5) генерація аналітичних звітів.

Переваги: гнучкий контроль витрат, інтуїтивний інтерфейс, можливість інтеграції з фінансовими установами. Недоліки: платна підписка, висока складність для новачків. Офіційний сайт: https://www.youneedabudget.com [17].

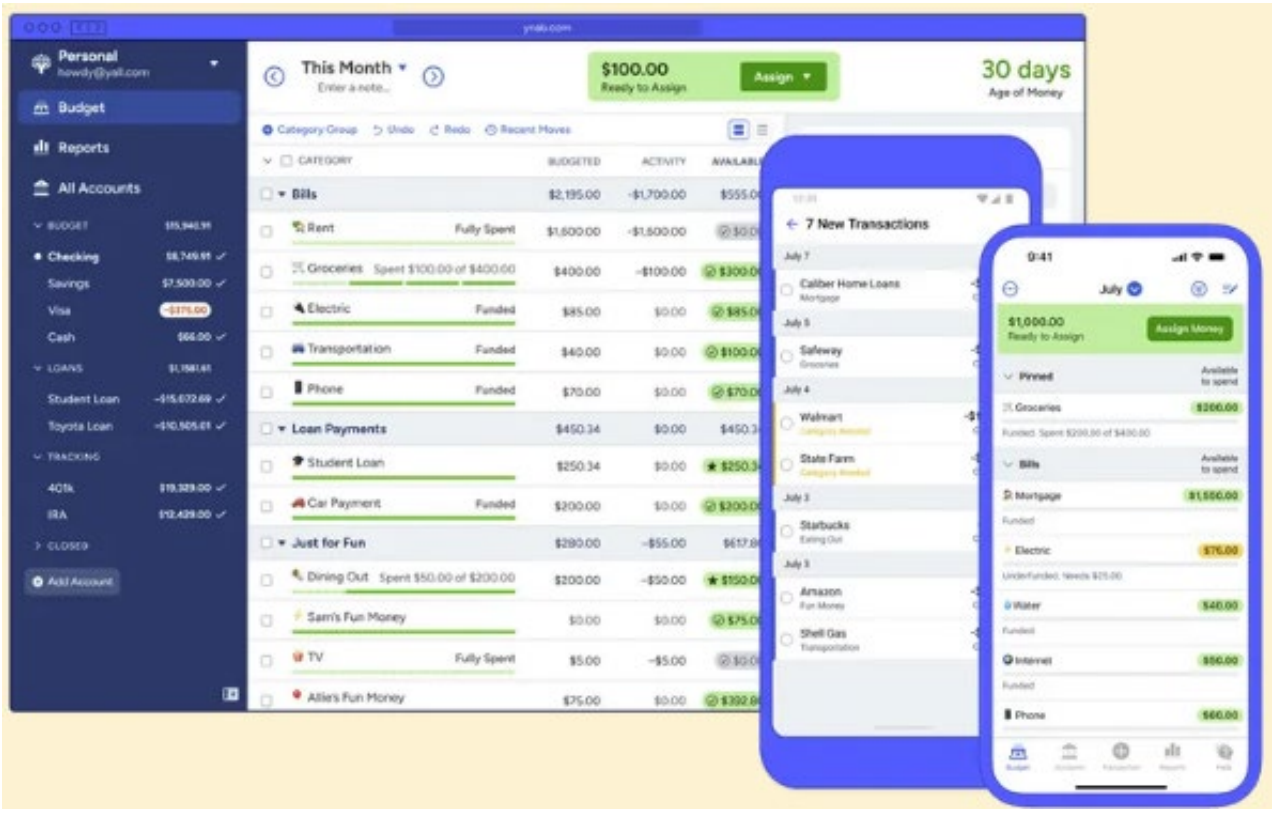

Рисунок 1.1 – інтерфейс YNAB (You Need A Budget)

Splitwise – розроблений компанією Splitwise Inc., цей мобільний та веб-додаток спеціалізується на розподілі витрат серед груп користувачів. Реалізований на Java та JavaScript, Splitwise є клієнт-серверним додатком, що

дозволяє користувачам легко фіксувати витрати та автоматично розраховувати борги. Застосунок підтримує синхронізацію даних у реальному часі, має простий та інтуїтивно зрозумілий інтерфейс, а також забезпечує зручне управління фінансами в групах, таких як друзі, родина чи колеги. Splitwise також інтегрується з платіжними системами, спрощуючи погашення заборгованостей між користувачами. Основні функції:

1) відстеження боргів і розподіл витрат;
2) автоматичні нагадування про платежі;
3) підтримка мультивалютності;
4) інтеграція з PayPal та Venmo;
5) історія всіх транзакцій та аналітика.

Переваги: простий у використанні, зручна функція розподілу витрат між користувачами, мобільний додаток з кросплатформенною підтримкою.

Недоліки: відсутність повноцінного бюджетування, обмежена функціональність без підписки. Офіційний сайт: https://www.splitwise.com [12].

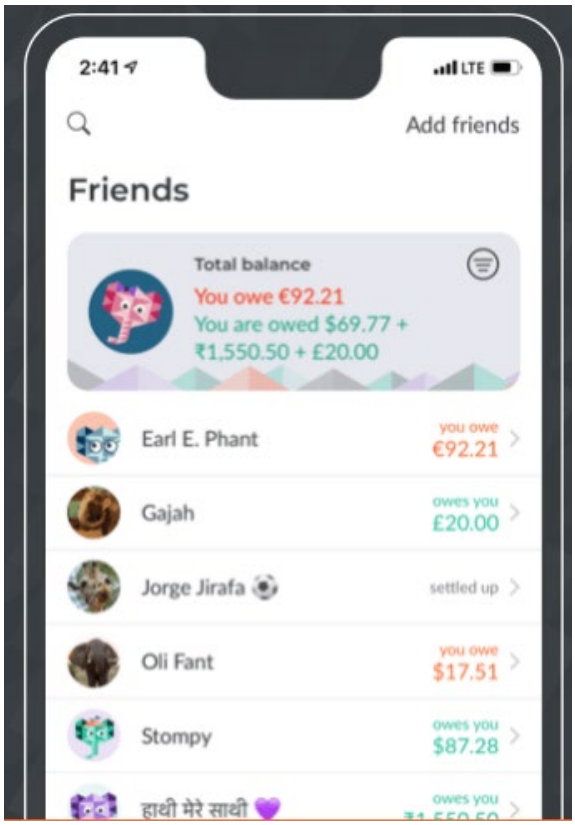

Рисунок 1.2 – інтерфейс Splitwise

Аналіз існуючих рішень показав, що вони мають певні недоліки, такі як обмежений функціонал, недостатня підтримка комплексного бюджетування та відсутність інтегрованих інструментів звітності. Це створює можливість для розробки більш гнучкої системи, яка поєднуватиме мультикористувацькі можливості, автоматизацію бюджетування та детальну фінансову аналітику.

Розробка нової системи повинна враховувати потреби як індивідуальних користувачів, так і груп, пропонуючи адаптивні механізми ведення бюджету, детальні звіти про фінансовий стан та високий рівень безпеки. Використання фреймворку Spring дозволить ефективно реалізувати серверну частину, забезпечивши продуктивність, масштабованість та інтеграцію з сучасними базами даних та засобами безпеки. Таким чином, запропоноване рішення буде відповідати сучасним вимогам до фінансових мобільних додатків та надасть користувачам ефективний інструмент для управління бюджетами [7].

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОЇ СИСТЕМИ БЮДЖЕТНОГО КОНТРОЛЮ З ФУНКЦІЯМИ ЗВІТНОСТІ ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ SPRING

2.1. Постановка задачі, призначення та вимоги до програмного засобу

Постановка задачі. Сучасна людина стикається з необхідністю контролю власних фінансів у режимі реального часу та на різних пристроях. Зростання кількості банківських карток, накопичувальних рахунків та регулярних платежів ускладнює ведення бюджетування без спеціалізованих інструментів. Існуючі рішення часто не забезпечують гнучкості у налаштуванні категорій витрат, не мають зручних механізмів локального кешування даних чи синхронізації між мобільним клієнтом і сервером. У зв’язку з цим постає потреба створити систему управління особистим бюджетом, яка поєднує надійний бекенд із гнучким Android-додатком, підтримує роботу в офлайн-режимі та гарантує безпечний обмін інформацією між користувачем і сервером.

Призначення системи. Розроблювана система покликана надати користувачеві цілісний інструмент для ведення, аналізу та планування особистих фінансів. Вона забезпечує реєстрацію та автентифікацію користувача, створення та редагування рахунків, облік різноманітних транзакцій із вказівкою категорії, суми й дати. Мобільний клієнт автоматично синхронізує зміни з сервером при наявності мережі, а також зберігає локальні копії даних для перегляду історії операцій в офлайн-режимі. Серверна частина реалізує надійний REST-інтерфейс із підтримкою JWT-токенів, відповідає за валідацію вхідних запитів, збереження ентіті в реляційній базі даних і відправлення підтверджень на електронну пошту. У результаті користувач отримує швидкий, зрозумілий та безпечний засіб контролю власного бюджету прямо з мобільного пристрою.

Вимоги до системи. До функціональних вимог належить можливість реєстрації нового користувача з підтвердженням e-mail, авторизації через JWT,

створення, редагування та видалення рахунків і транзакцій із зазначенням категорій та приміток. Система повинна підтримувати перегляд зведених звітів за вибраний період, фільтрацію операцій за категоріями й датами, а також автоматичне оновлення даних при зміні статусу мережі. Нефункціональні вимоги охоплюють забезпечення конфіденційності (шифрування паролів, захист токенів), продуктивності (час відповіді API до 300 мс під навантаженням до 100 запитів на секунду), зручності інтерфейсу (відповідність гайдлайнам Android, адаптивність під різні розміри екранів), стійкості до збоїв (автоматичне повторне підключення клієнта, транзакційна цілісність на сервері) та масштабованості (можливість розгортання в хмарі й розширення бази даних без простоїв).

Технічне завдання описане в Додатку А.

2.2. Вибір моделі розробки програмного засобу

Життєвий цикл цього програмного продукту охоплює послідовність фаз від планування до виведення з експлуатації згідно з ISO / IEC 12207. На стадії планування і з’ясування вимог формується повний перелік функціональних можливостей — від налаштування бюджету й обліку транзакцій до побудови звітів і взаємодії з сервером через REST інтерфейс.

Після затвердження вимог переходять до проєктування архітектури, де вибирається клієнт серверна модель: серверна частина на Spring Boot відповідає за зберігання й обробку даних у PostgreSQL, а мобільний клієнт на Kotlin керує інтерфейсом та локальним кешем. Така побудова гарантує розширюваність і ефективне передавання інформації.

На стадії розробки безпосередньо пишеться код для обох частин системи із суворим дотриманням специфікацій. Серверна логіка реалізується через контролери й сервіси Spring, клієнт — у вигляді MVVM структури на Android. Особлива увага приділяється коректній взаємодії з базою даних і надійному збереженню транзакцій.

Після завершення кодування проводиться комплексне тестування: перевіряють правильність роботи API, стійкість мобільного інтерфейсу й взаємодію між клієнтом і сервером. Автоматизовані тести на JUnit забезпечують регресійне покриття ключових сценаріїв.

Впровадження відбувається при задокументованому успішному проходженні всіх перевірок: система розгортається на продакшн сервері, мобільний застосунок публікується в магазині, налаштовуються засоби моніторингу та підтримки.

У фазі експлуатації здійснюють постійний моніторинг працездатності, усувають виявлені дефекти й випускають оновлення відповідно до зворотного зв’язку від користувачів. Коли система втрачає актуальність або готується її нова версія, проводять процедуру виведення з експлуатації, що передбачає архівацію даних і перенесення користувачів на оновлену платформу.

Для реалізації цього проєкту було обрано класичний водоспадний підхід, оскільки на початку робіт вимоги до функціональності та якості були чітко визначені й не передбачалося їх суттєвої зміни. Послідовне виконання фаз гарантує ретельне планування, документування та перевірку кожного етапу, що особливо важливо для фінансової системи з високими вимогами до надійності й точності обліку .

2.3. Архітектура та структура застосунку

У ході виконання дипломної роботи створено клієнт-серверний застосунок, в якому мобільний клієнт реалізовано на Kotlin для Android, а серверна частина побудована з використанням Java та Spring Boot. Для довготривалого зберігання інформації обрано реляційну базу даних PostgreSQL

Архітектура системи складається з трьох рівнів. Перший рівень відповідає за взаємодію з користувачем і представлення даних у вигляді екранних форм та елементів керування. Другий рівень містить бізнес-логіку застосунку: обробку запитів, валідацію, координування операцій між клієнтом і базою даних. Третій

рівень реалізує роботу з базою даних, забезпечуючи збереження, вибірку та оновлення записів через JPA/Hibernate. Така організація дозволяє ізольовано розвивати інтерфейс, логіку й шар зберігання даних, спрощує тестування та підтримку, а також гарантує можливість масштабувати кожен із компонентів незалежно один від одного.

2.3.1. Серверна частина на базі Spring Boot

Серверна частина поділена на функціональні модулі. У модулі application зосереджені REST контролери й сервіси, які координують роботу інших компонентів. Модуль validators забезпечує перевірку вхідних даних, а domain містить JPA сутності та репозиторії для роботи з базою. Відповіді та запити клієнта формуються в модулі presentation, а загальні допоміжні класи — наприклад, для обробки JWT токенів чи шифрування паролів — розміщені в util. Безпекові налаштування й інтеграція Spring Security виконані в модулі infrastructure.

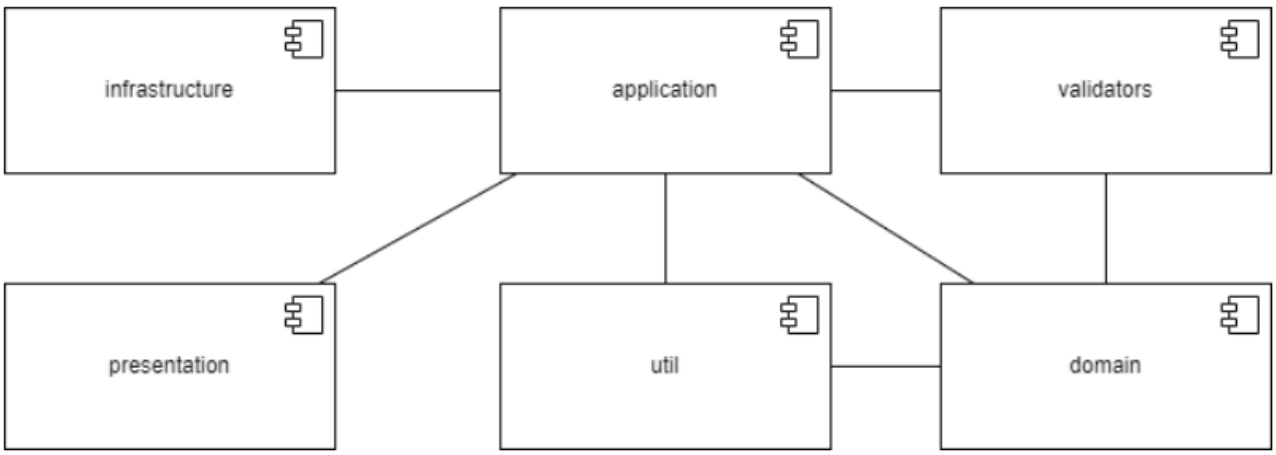

Рисунок 2.1 – Архітектура серверу

Сервер побудований як автономний Spring Boot-додаток із трирівневою архітектурою: REST-контролери приймають HTTP-запити, сервіси виконують бізнес-логіку, репозиторії взаємодіють із PostgreSQL через JPA/Hibernate. Приклад методу в контролері, що створює нову транзакцію, демонструє суть обробки запиту і повернення відповіді:

```
@PostMapping("/transactions")
public ResponseEntity<TransactionDto> createTransaction(
    @RequestBody @Valid TransactionDto dto,
    @AuthenticationPrincipal UserDetails user
) {
    TransactionDto created = transactionService.create(dto, user.getUsername());
    return ResponseEntity.status(HttpStatus.CREATED).body(created);
}
```

У цьому фрагменті валідація запиту відбувається автоматично завдяки анотації @Valid, а інформація про користувача дістається з контексту безпеки.

Конфігурація безпеки зосереджена в класі, що розширює WebSecurityConfigurerAdapter. Там визначено фільтр JWT-токенів, схему кодування паролів і правила доступу до шляхів API. Нижче наведено уривок з налаштування без стану (stateless) та додаванням власного JwtAuthFilter:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .authorizeRequests()
            .antMatchers("/auth/**").permitAll()
            .anyRequest().authenticated()
        .and()
        .addFilterBefore(jwtAuthFilter(),
UsernamePasswordAuthenticationFilter.class);
}
```

Цей підхід гарантує, що жодна частина API не залишиться необережено відкритою, а перевірка прав проводиться на рівні HTTP-фільтрів до виклику контролерів.

Шари сервісів побудовані навколо транзакцій Spring, що дозволяє забезпечити атомарність операцій із базою. Наприклад, метод створення рахунку міститься в AccountService, а його декларація виглядає так:

```
@Transactional
public AccountDto createAccount(AccountDto dto, String username) {
    User user = userRepository.findByEmail(username)
        .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    Account entity = accountMapper.toEntity(dto);
    entity.setUser(user);
    return accountMapper.toDto(accountRepository.save(entity));
}
```

Цей код ілюструє, як бізнес логіка, обробка помилок та перетворення між DTO і Entity залишаються чітко розділеними.

2.3.2. Клієнтська частина: мобільний додаток на Kotlin

Архітектура Android-клієнта побудована за принципом розділення на шари Presentation, Domain і Data.

У шарі Presentation розміщені елементи інтерфейсу (View) та їх ViewModel. View реагує на дії користувача — натискання кнопок, введення тексту, жестові події — і передає їх у ViewModel. ViewModel зберігає актуальний стан екрану у вигляді observable-полів, ініціює запуск відповідних UseCase із шару Domain і стежить за їхнім виконанням. Саме ViewModel формує ті дані, які відображаються на екрані, перетворюючи внутрішні моделі на UI-дружні об’єкти. Завдяки розділенню на View та ViewModel інтерфейс може змінюватися без втручання в бізнес-логіку, а навпаки — внутрішня логіка доповнюватися без правок у макетах і анімаціях.

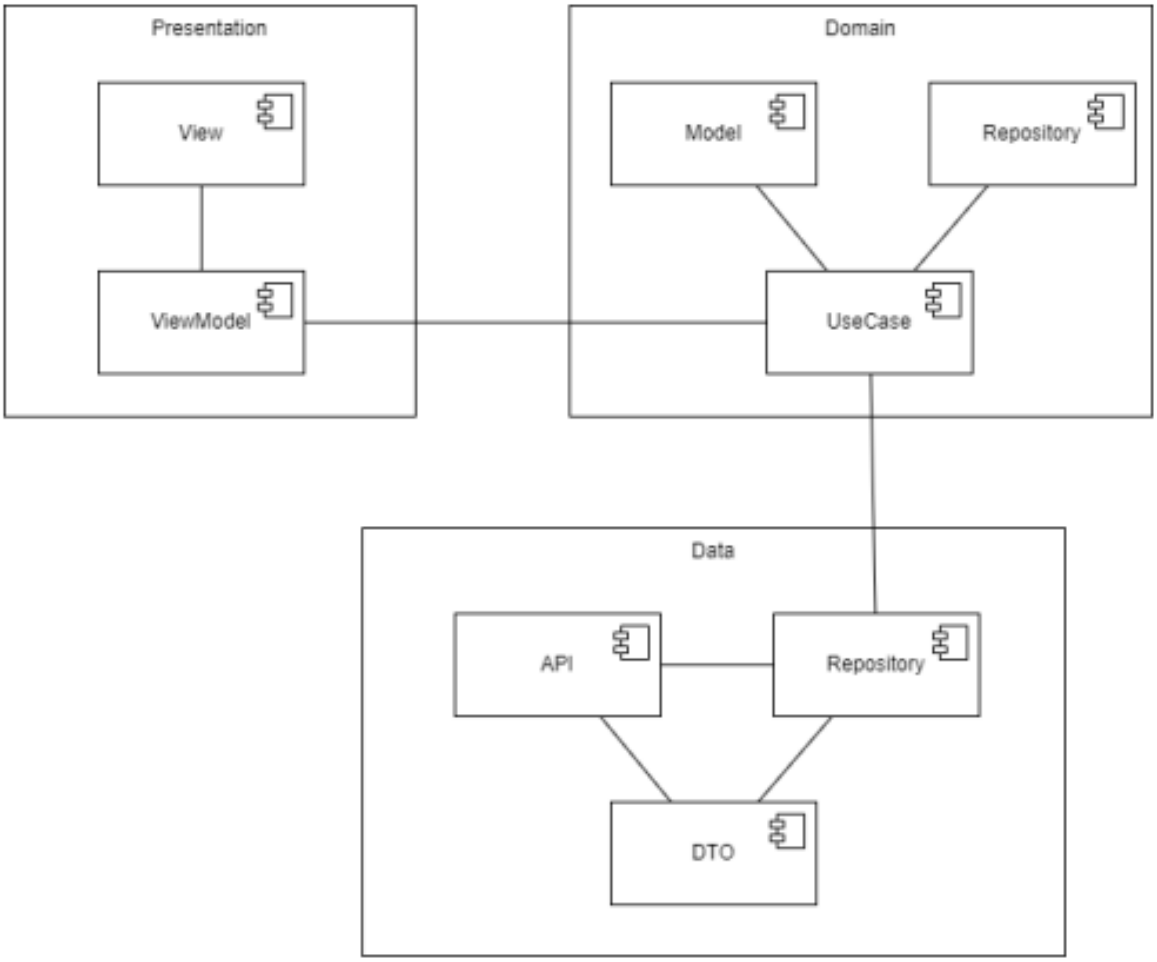

Рисунок 2.2 – Архітектура клієнту

Domain-шар зосереджує бізнес-правила у вигляді UseCase. Кожен UseCase відповідає за один сценарій використання (наприклад, отримати список транзакцій, створити новий рахунок, виконати авторизацію). UseCase звертається до Repository-інтерфейсів, не знаючи деталей їх реалізації, і повертає результат у вигляді Model-об’єктів. Model описує сутності предметної області (транзакція, рахунок, користувач) із відповідними властивостями й методами перевірки цілісності. Така ізоляція гарантує, що будь-які зміни в способі збереження або мережевій взаємодії не потребуватимуть переробки бізнес-логіки.

Шар Data реалізує інтерфейси репозиторіїв, забезпечуючи роботу з двома джерелами даних: віддаленим REST-API і локальною базою Room. Retrofit-клієнти перетворюють HTTP-відповіді на DTO, які потім мапляться у

внутрішні Model або зберігаються в SQLite через DAO-класи. Коли мережа недоступна, репозиторій повертає кешовані записи, що дозволяє застосунку працювати в офлайн-режимі. При відновленні з'єднання відбувається автоматична синхронізація: нові локальні зміни надсилаються на сервер, а оновлені дані завантажуються назад.

Комунікація між шарами здійснюється за чітко визначеними контрактами: ViewModel викликає UseCase, UseCase користується Repository, Repository — API або DAO. На рисунку 2.2 показано ці взаємозв'язки: стрілки вказують напрямок викликів, а прямокутники — відповідні компоненти кожного шару.

Завдяки такій структурі впровадження нових функцій, наприклад, додавання графічного аналізу витрат або підтримки мультивалютності, зводиться до створення нових UseCase та розширення відповідних репозиторіїв, без необхідності змінювати UI-логіку або низькорівневий мережевий код. Це пришвидшує розробку, знижує кількість помилок і полегшує підтримку застосунку в процесі його еволюції.

2.3.3. Архітектура та реалізація бази даних на PostgreSQL

Архітектура бази даних побудована на PostgreSQL із урахуванням потреб клієнт-серверної системи бюджетного контролю. Дані організовано в реляційні таблиці, які відображають предметну область: користувачі, їхні рахунки, транзакції та категорії витрат. При цьому застосовано нормалізацію до третьої нормальної форми, щоб уникнути надлишкового дублювання й забезпечити цілісність зв'язків.

Кожен запис користувача зберігає ідентифікатор, email, хеш пароля й дату реєстрації. Рахунок прив'язаний до користувача через зовнішній ключ і містить назву, тип (наприклад, готівка чи картка) і поточний баланс. Транзакція зв'язується з певним рахунком і категорією, фіксує суму, дату-час операції та текстову примітку. Категорії визначаються одним набором стандартних значень (їжа, транспорт тощо) і зберігаються окремо, щоб забезпечити довільне розширення переліку.

Конструкції FOREIGN KEY і ON DELETE CASCADE гарантують, що при видаленні користувача автоматично ліквідуються його рахунки та пов’язані транзакції, натомість категорії зберігаються незалежно від змін у користувацьких записах. Індекси на полях user_id, account_id і category_id прискорюють вибірки звітів за період чи по категорії, а часткові індекси на даті операції оптимізують вибірку останніх транзакцій.

Для візуалізації структури наведено ER-діаграму (Рисунок 2.3), де прямокутники позначають сутності, ромби – зв’язки, а підписи біля ліній вказують кардинальність.

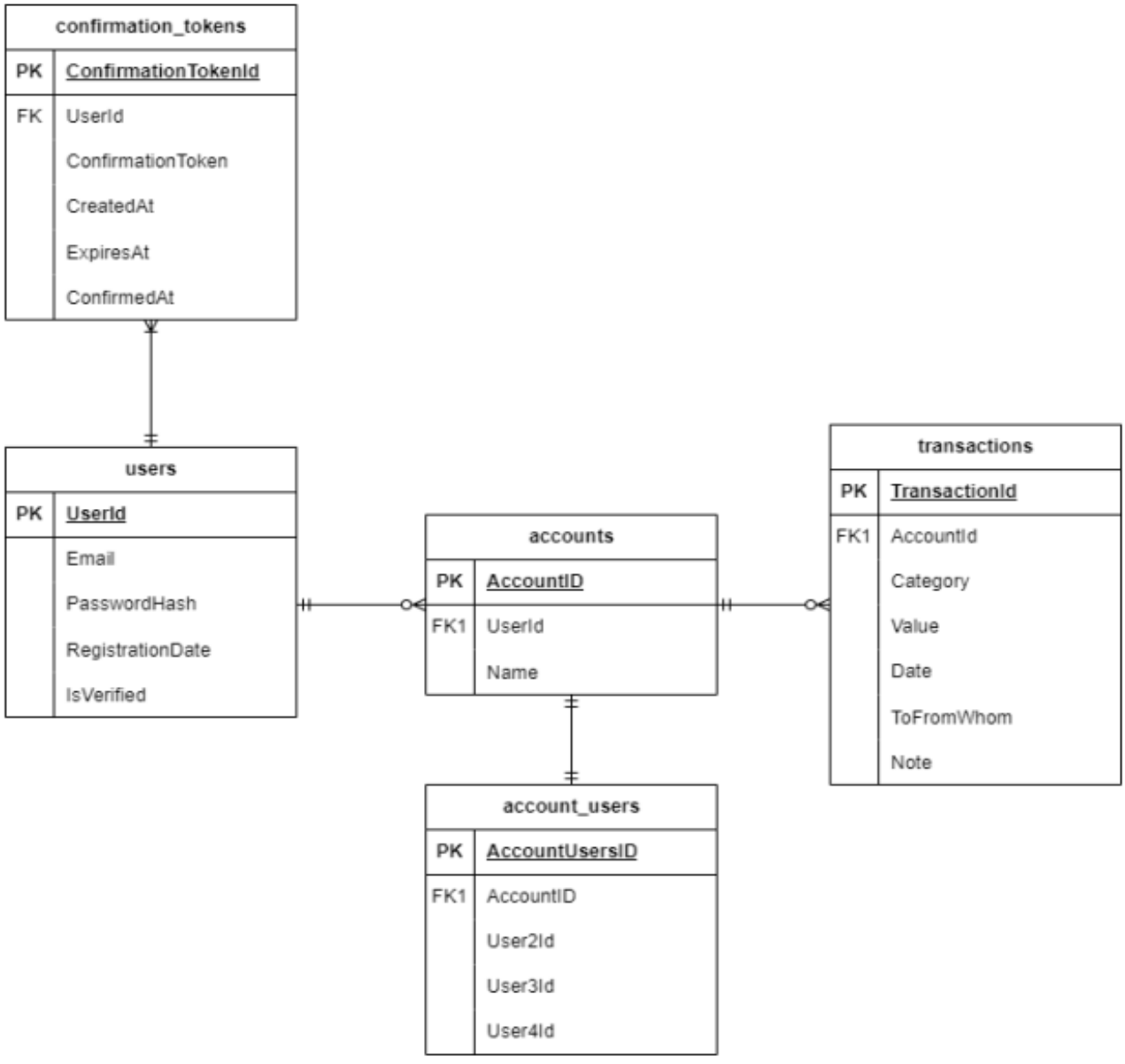

Рисунок 2.3 – ER-модель бази даних PostgreSQL

Таким чином, побудована схема забезпечує гнучкість розширення, швидкість запитів і збереження цілісності даних у фінансовій системі.

2.4. Обґрунтування вибору технологій та інструментів для системи бюджетного контролю

Для реалізації мультикористувацької мобільної системи бюджетного контролю було обрано технологічний стек, який максимально відповідає вимогам продуктивності, масштабованості та зручності підтримки. В основі клієнтської частини лежить Android-платформа з використанням Kotlin, що гарантує високу швидкість розробки та стабільність виконання на широкому діапазоні пристроїв. З точки зору серверного боку ключовим компонентом став Spring Boot, який забезпечує гнучку конфігурацію REST-інтерфейсів, надійність механізмів безпеки та легку інтеграцію з базою даних за допомогою Spring Data JPA.

У виборі клієнтських бібліотек перевага віддана Jetpack-компонентам, які спрощують управління життєвим циклом екранів і дозволяють вбудувати в додаток архітектуру MVVM. Використання ViewModel та LiveData сприяє зменшенню складності при обробці даних та полегшує написання модульних тестів. Для організації залежностей обрано Hilt – офіційне рішення від Google, котре інтегрується з Android Studio й надає простий API для впровадження залежностей та їхньої автоматичної ін’єкції. Мережевий рівень реалізований через Retrofit в поєднанні з OkHttp, що дає можливість оптимізувати запити та забезпечити обробку помилок на етапі HTTP-клієнта. Локальне зберігання даних побудовано на основі Room, який спрощує роботу з SQLite та забезпечує автоматичну перевірку запитів під час компіляції.

Серверна частина спроектована з урахуванням принципів «конфігурація замість коду», що реалізовано через Spring Boot. Завдяки автоматичному скануванню компонентів та вбудованим механізмам автоконфігурації зводиться до мінімуму обсяг рукописної конфігурації. REST-ендпоїнти організовано за

допомогою Spring Web MVC, що дає можливість просто визначати контролери та обробляти HTTP-запити. Усі сервіси поділено на шари контролерів, сервісів бізнес-логіки та шар доступу до даних, що реалізовано через Spring Data JPA. Використання Hibernate у ролі ORM-провайдера гарантує сумісність із різними СУБД, а рівень абстракції Spring Data полегшує розробку репозиторіїв.

Для зберігання фінансових даних обрано реляційну базу PostgreSQL, котра вирізняється стабільністю, підтримкою транзакційних операцій та потужними інструментами для резервного копіювання. PostgreSQL здатна обробляти великі обсяги даних і водночас надає можливості оптимізації запитів через індекси та планувальник виконання. Конфігурація підключення виконується через пул HikariCP, інтегрований у Spring Boot, що забезпечує ефективне керування з’єднаннями та низьку затримку під час пікових навантажень.

Питання безпеки системи вирішено на основі Spring Security у поєднанні з JWT-маршрутизацією токенів. Такий підхід виключає необхідність зберігання сесій на сервері, дозволяючи клієнтам самостійно передавати токен у заголовках запитів. Це спрощує масштабування мікросервісів і дає змогу чітко регламентувати права доступу через опис аутентифікаційних фільтрів та політик браузер-дружніх CORS-настройок.

При розробці системи звітності було застосовано бібліотеку JasperReports для серверного генерування PDF-звіту, що надає змогу формувати готові до вивантаження документи з динамічним наповненням таблиць і графіків. Одночасно клієнтська частина підтримує візуалізацію даних за допомогою MPAndroidChart, що дозволяє відображати інтерактивні діаграми та гістограми без додаткових серверних запитів.

Контейнеризація окремих модулів реалізована через Docker, що полегшує розгортання системи у різних середовищах та гарантує ідентичність конфігурацій під час тестування та продакшену. Збірку артефактів організовано через Gradle Kotlin DSL із чітким поділом на скрипти клієнта та сервера. Розгортання на CI/CD-серверах автоматизовано за допомогою GitHub Actions, де

налаштовано перевірку статичного аналізу коду, запуск тестів та публікацію артефактів у приватний репозиторій.

Таким чином, обрані технології та інструменти створюють єдине високопродуктивне середовище розробки та експлуатації мобільної системи бюджетного контролю. Поєднання Kotlin і Android Jetpack на фронтенді з Spring Boot і PostgreSQL на бекенді забезпечує легкість підтримки, відповідність сучасним вимогам безпеки й простоту масштабування разом із зручними засобами звітності для кінцевих користувачів.

2.5. Особливості програмної реалізації системи бюджетного контролю

Перед описом ключових модулів реалізації варто окреслити загальний підхід до побудови архітектури та взаємодії компонентів. У системі бюджетного контролю клієнт і сервер тісно співпрацюють за допомогою чітко визначених REST-ендпоїнтів, що дозволяє розділити відповідальність між шаром представлення, бізнес-логікою та збереженням даних. Такий підхід гарантує гнучкість масштабування та полегшує тестування окремих частин застосунку. Нижче наведено основні функціональні блоки з детальним описом їх реалізації.

Автентифікація та авторизація користувачів. У проєкті реалізовано безперервний процес автентифікації через REST-ендпоінт /api/auth/login, який приймає облікові дані користувача та повертає пару токенів – короткостроковий JWT для доступу до ресурсів та довгостроковий refresh-токен. Сервер на Spring Boot перевіряє логін і пароль у базі даних через Spring Security, створює JWT з наборами ролей у полі claims та встановлює строк дії в 15 хвилин, тоді як строк refresh-токена сягає 7 діб. На клієнті Android використовується OkHttp Interceptor, який автоматично додає токен у заголовок Authorization при кожному запиті до захищених ендпоінтів. У разі закінчення строку дії access-токена відбувається виклик /api/auth/refresh, що дозволяє заощадити користувачеві повторний ввід пароля. Механізми авторизації побудовано на фільтрах Spring Security, кожен запит проходить перевірку ролей у токені.

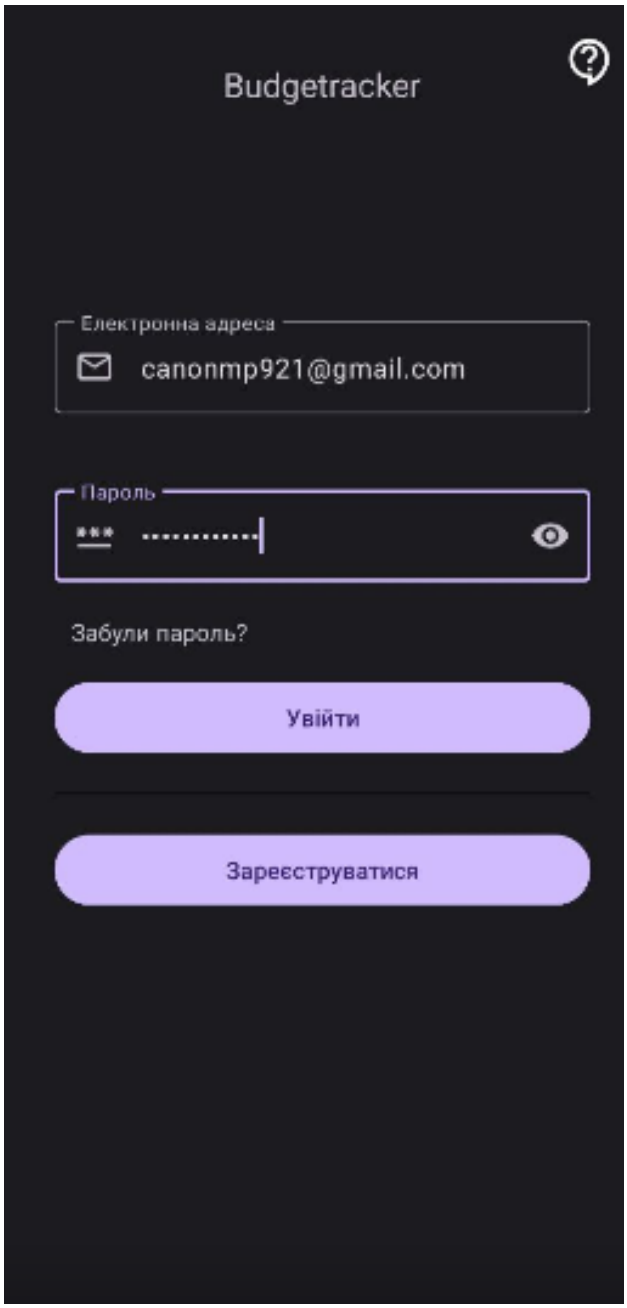

Рисунок 2.4 – Сторінка авторизації користувача

Управління доходами та витратами. В основу функціоналу покладено єдину сутність Transaction, що містить поля для суми, дати, категорії та опису. Серверна частина на Spring Data JPA обробляє CRUD-операції з цією сутністю через репозиторії, а клієнтська частина у ViewModel передає об’єкти в локальну базу Room для офлайн-доступу. Синхронізація відбувається асинхронно: зміни накопичуються у черзі та передаються на сервер у фоновому режимі. У випадку

конфлікту версій застосовується стратегія останнього запису та повідомлення користувачу про необхідність ручного узгодження.

Генерація фінансових звітів. Формування звітів реалізовано на серверній стороні через JasperReports, що дозволяє підставляти дані з бази у заздалегідь підготовлені шаблони. Користувач обирає період у клієнтському додатку, після чого відповідний запит передається на бекенд, де генерується PDF-документ. Докладний звіт містить загальні підсумки за категоріями, графічні елементи та текстовий аналіз. Згенерований файл повертається клієнту у відповіді та зберігається у файлах програми для подальшого перегляду, завантаження або пересилки електронною поштою.

Рисунок 2.5 – Звіт у форматі PDF

Візуалізація даних та графіки. Для інтерактивного відображення дашборду на клієнті використано MPAndroidChart, що підтримує стовпчикові діаграми. Дані для графіків запитуються з REST-ендпоїнтів у форматі JSON, у якому передаються агреговані значення за обраними категоріями та часом. ChartView автоматично оновлює відображення при зміні вхідних даних, а вигляд діаграм кастомізовано через XML-стилі Android. Завдяки оптимізації оновлення у RecyclerView загальна продуктивність інтерфейсу залишається стабільною навіть для великої кількості точок даних.

Керівництво користувачу наведено в Додатку Б

2.6. Організація тестування та налагодження програмного засобу

Тестування серверної частини починається з написання модульних тестів для всіх компонентів бізнес-логіки та доступу до даних. Застосунок використовує JUnit 5 і Mockito для імітації залежностей, що дозволяє перевіряти методи сервісів і репозиторіїв без підключення до реальної бази даних. Інтеграційні тести на Spring Boot Test піднімають контекст додатка в ізольованому середовищі, де за допомогою Testcontainers розгортається тимчасовий контейнер PostgreSQL. У кожному запуску перевіряються REST-ендпоїнти через MockMvc, зокрема сценарії успішної і невдалої автентифікації, управління транзакціями і генерація звітів.

Тестування Android-клієнта організовано на двох рівнях: юніт-тести ViewModel і допоміжних класів виконує Robolectric у JVM-середовищі для швидкої перевірки логіки без емулятора. UI-тести написані за допомогою Espresso і виконуються на емуляторах та фізичних пристроях у різних конфігураціях Android API. Кожен сценарій покриває процеси входу, створення й редагування транзакцій, перегляд графіків із MPAndroidChart та відображення PDF-звіту у вбудованому переглядачі. При виникненні нестабільності в UI-тестах використовується idling resources для синхронізації з фоновими

процесами, а також екрани з дампом логів Android Logcat, що допомагає локалізувати причину залежання чи невідповідності інтерфейсу.

Для перевірки взаємодії клієнта та сервера застосовується набір тестів на базі REST-assured і Restman. Цей інструментарій виконує сценарії повного циклу, починаючи від реєстрації користувача, отримання токена, створення кількох транзакцій, запиту звітів і закінчуючи очищенням тестових даних. Результати цих тестів публікуються у вигляді HTML-звіту.

Налагодження серверних компонентів реалізоване через Logback з поділом логів за рівнями DEBUG, INFO, WARN і ERROR. Кожен ключовий сервіс веде власний лог-файл, що дозволяє швидко відслідковувати виконання запитів, вибірку даних і помилки Hibernate. Для аналізу продуктивності застосований JVisualVM та Flight Recorder, які допомагають виявляти вузькі місця в роботі пулу з’єднань HikariCP і в алгоритмах формування звітів.

Загальна стратегія тестування і налагодження забезпечує багатоетапний контроль якості, що гарантує стабільність роботи програми під різними умовами та швидке виявлення і виправлення дефектів на будь-якому етапі розробки.

2.7. Рекомендації щодо використання та впровадження мобільного додатка

Для розгортання серверної частини на Windows 10 або Windows 11 необхідно попередньо встановити PostgreSQL (версії 13 чи новішої), графічний клієнт pgAdmin, Git та Android SDK. Після інсталяції слід запустити службу PostgreSQL через стандартний інструмент «Служби» Windows: знайти запис із назвою postgresql-x64-13 (або подібну, відповідно до версії) і вибрати команду «Запустити».

Далі візьміть pgAdmin і підключіться до локального сервера на порті 5432, використовуючи облікові дані адміністратора (за замовчуванням — користувач postgres і ваш пароль). Далі потрібно створити нову базу дану для програмного продукту, це можна зробити за допомогою наступної команди:

«CREATE DATABASE budgetappdb;».

Після чого треба перейти до папки server/src/main/resources, де слід створити файл application.properties з параметрами доступу до бази та вказати налаштування SMTP-пошти (рисунок 2.6).

```
spring.datasource.url=jdbc:postgresql://localhost:5432/budgetappdb
spring.datasource.username=postgres
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update

spring.security.secret-key=357638792F423F4428472B4B6250655368566D597133743677397A2443264629

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=test@gmail.com
spring.mail.password=password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

properties.address=http://localhost:8080
```

Рисунок 2.6. – Файл application.properties

Збережений конфігураційний файл стане основою для запуску Spring Boot-додатка. Тоді треба повернутися до директорії server і запустити сервер командою .\gradlew bootRun (рисунок 2.7).

Наступним кроком слід оновити IP-адресу сервера в налаштуваннях мобільного додатку. Для цього відкрийте папку *client/app/src/main/java/com/vomelianchuk/app/budgettracker/client/common* і внесіть потрібні зміни у файл Constants.kt (рисунок 2.8).

Після цього у папці client виконайте команду gradlew assembleDebug, щоб зібрати та отримати вихідний APK у режимі налагодження. Отриманий файл можна встановити на Android-пристрій через ADB або Android Studio, після чого додаток готовий до роботи.

Для більш ефективного використання та підтримки системи рекомендується виконати наступні дії.

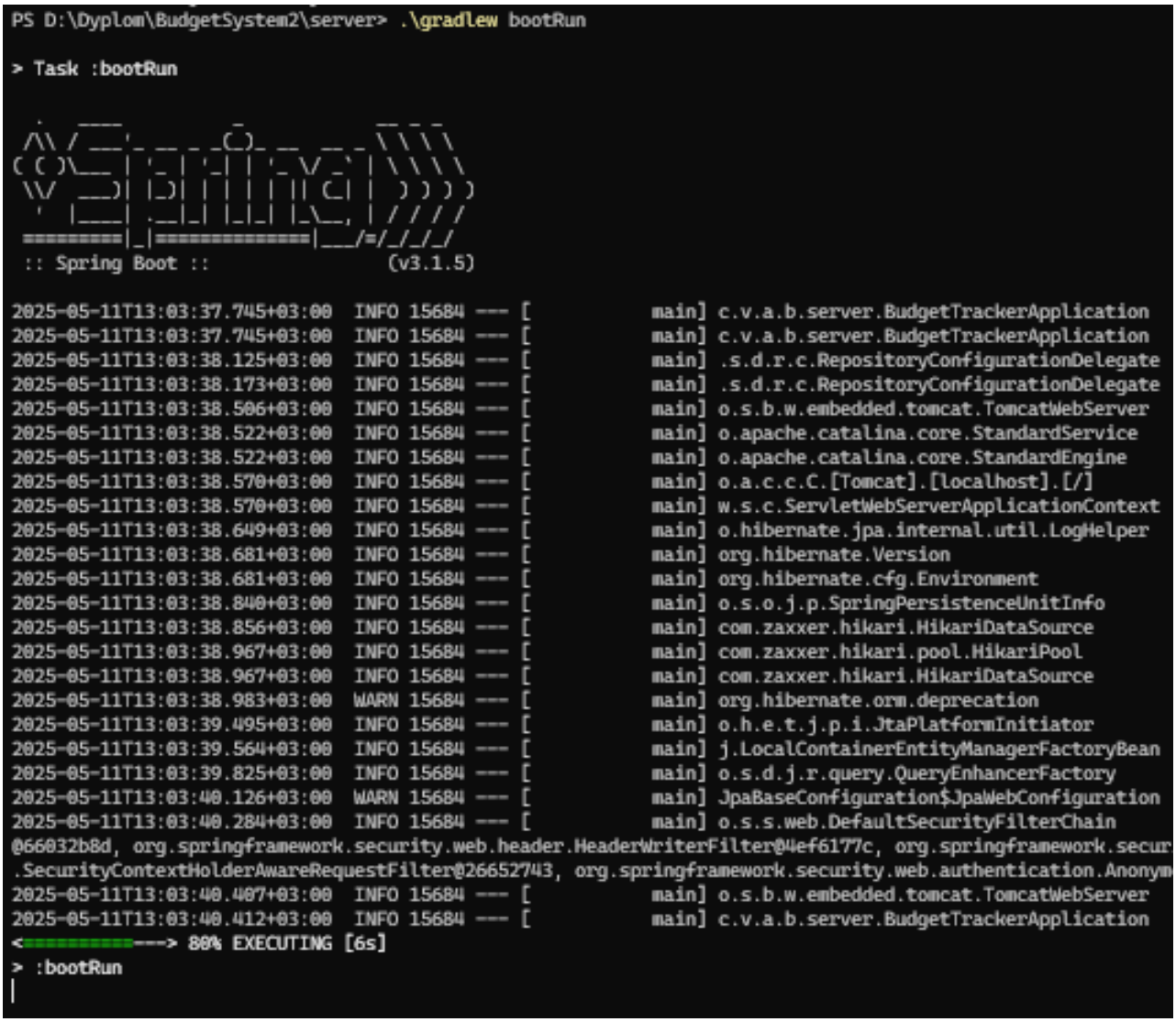

Рисунок 2.7 – Запуск серверу

```
object Constants {
    /**
     * Name of app preferences
     */
    const val PREFERENCES = "Budgetracker"

    /**
     * Object with API URLs
     */
    object URL {
        const val BASE_URL = "http://192.168.31.195:8080/"
        const val AUTH_URL = "api/v1/auth/"
        const val USER_URL = "api/v1/users/"
        const val ACCOUNT_URL = "api/v1/accounts/"
        const val TRANSACTION_URL = "api/v1/transactions/"
    }
```

Рисунок 2.8 – Зміна налаштувань мобільного застосунку

По-перше, налаштувати регулярне резервне копіювання бази даних PostgreSQL із автоматичним збереженням дампів на окремому диску або мережевому сховищі. Це дозволить швидко відновити роботу сервера після будь-яких збоїв.

По-друге, забезпечити моніторинг запущеного додатка за допомогою вбудованих метрик Actuator або зовнішніх сервісів (наприклад, Prometheus + Grafana), щоб відслідковувати навантаження, час відповіді й використання ресурсів.

По-третє, періодично оновлювати залежності клієнтської та серверної частини, проводячи прогін тестів після кожного оновлення через CI/CD-конвеєр на GitHub Actions, аби вчасно виявляти потенційні конфлікти чи вразливості.

По-четверте, організувати короткі навчальні сесії для кінцевих користувачів із демонстрацією інтерфейсу додатка: показати, як правильно створювати транзакції, переглядати графіки та генерувати PDF-звіт.

І нарешті, встановити систему підтримки помилок і запитів, щоб збирати відгуки й оперативно реагувати на питання користувачів.

Дотримання цих рекомендацій значно підвищить надійність, безпеку та зручність експлуатації мобільного додатка бюджетного контролю.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було створено повноцінний програмний продукт для контролю особистого та сімейного бюджету, що складається з десктопної серверної частини на основі Spring Boot і реляційної бази даних PostgreSQL та мобільного Android-додатка на Kotlin. Завдяки реалізації клієнт-серверної архітектури користувачі отримали змогу працювати зі своїми фінансовими даними у режимі реального часу, користуючись зрозумілим інтерфейсом на мобільному пристрої. Сервер обробляє запити, управляє збереженням і цілісністю даних, надає механізми аутентифікації й авторизації на основі JWT, а клієнт гарантує зручність введення транзакцій, налаштування категорій доходів і витрат та відображення звітів.

Розроблене рішення спростило облік доходів і витрат через єдину структуру даних і автоматичну синхронізацію між локальним кешем і сервером. Використання Spring Data JPA дало можливість швидко створювати надійні репозиторії для операцій з базою даних, а Android Room забезпечив безперебійну роботу застосунку в умовах нестабільного інтернет-з’єднання. Генерація звітів за допомогою JasperReports на боці сервера дозволяє формувати докладні PDF-документи з графіками й текстовими висновками.

Безпеку системи забезпечують багаторівнева перевірка токенів і контроль швидкості запитів. У поєднанні зі стандартними практиками резервного копіювання PostgreSQL та моніторингом через Spring Actuator це створює стійке середовище, готове витримувати навантаження й автоматично повідомляти про критичні події.

Розроблений мобільний адаптивний інтерфейс підтримує Android 8.0 і вище, що охоплює велику кількість сучасних пристроїв. Використання MPAndroidChart для візуалізації дашборду дозволяє відображати динаміку доходів і витрат у вигляді стовпчикових діаграм, а Cache та DI-фреймворк Hilt оптимізують продуктивність і роботу з пам’яттю на клієнтській стороні.

Водночас проект має потенціал для подальшого розвитку: інтеграція з платіжними шлюзами й зовнішніми обліковими системами підвищить гнучкість у корпоративному середовищі, а впровадження модулів аналітики на базі машинного навчання дозволить прогнозувати фінансові тренди й рекомендовані стратегії заощаджень. Також доцільним є розширення підтримки інших платформ, зокрема iOS, та додавання багатомовного інтерфейсу для користувачів із різних регіонів.

Отже, реалізована система продемонструвала свою здатність покращувати процес бюджетного контролю, об’єднавши сучасні технології серверної та мобільної розробки. Вона забезпечує простий і безпечний доступ до фінансових даних, гнучку генерацію звітів і масштабовану архітектуру, готову до розвитку й удосконалення відповідно до змінних потреб кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler, M. (2002). Patterns of Enterprise Application Architecture. – Addison-Wesley.
2. Goodbudget. Офіційний сайт. URL: https://goodbudget.com
3. Johnson R., Hoeller J. Spring Boot in Action. Manning Publications, 2017.
4. Johnson, R., Hoeller, J. (2020). Spring Framework Documentation. – Spring.io. https://docs.spring.io/spring-framework/docs/current/reference/html/
5. Kotenko, I., & Chechulin, A. (2017). Security of Information and Communication Systems: Methods, Models and Solutions. – Springer.
6. Makarychev, A. (2021). Spring Security in Action. – Manning Publications.
7. Mockito Team. Official Mockito Documentation. [online]. Available: https://site.mockito.org/
8. Müller M. JUnit 5 User Guide. JUnit.org, 2021.
9. PostgreSQL Documentation. Офіційний сайт. URL: https://www.postgresql.org/docs/
10. PostgreSQL Global Development Group. (2023). PostgreSQL Documentation. https://www.postgresql.org/docs/
11. Richardson, C. (2018). Microservices Patterns: With examples in Java. – Manning Publications.
12. Splitwise. Офіційний сайт. URL: https://www.splitwise.com
13. Spring Framework. Офіційна документація. URL: https://spring.io
14. Spring Security & JWT Authentication. Офіційна документація. URL: https://spring.io/projects/spring-security
15. Syrota, T., & Melnyk, A. (2022). Інформаційна безпека в сучасних інформаційних системах // Наукові записки НУК. – №5. – С. 22–29.
16. Walls, C. (2016). Spring Boot in Action. – Manning Publications.
17. You Need A Budget. Офіційний сайт. URL: https://www.youneedabudget.com

18. Ларіонова І. І., Ситник Г. І. Фінансові технології та електронні платежі: сучасний стан та перспективи розвитку // Економічний простір. – 2021. – №177. – С. 89–98.
19. Марченко, О. В. Порівняльний аналіз функціоналу сучасних мобільних додатків для бюджетного планування // Управління розвитком складних систем. – 2023. – № 43. – С. 65–70.
20. Протасова Т. В. Особливості функціонування фінансових мобільних додатків у цифрову епоху // Бізнес Інформ. – 2020. – № 6. – С. 127–132.

ДОДАТКИ

Додаток А.

Технічне завдання

Вступ

Програмний продукт, що розробляється, має назву «Мультикористувацька мобільна система бюджетного контролю з функціями звітності за допомогою фреймворку Spring». Він призначений для використання в галузі управління особистими та спільними фінансами, надаючи користувачам інструменти для відстеження доходів і витрат, планування бюджетів, генерації звітів та співпраці з іншими користувачами у питаннях спільного фінансового менеджменту. Програма розробляється для мобільних пристроїв, зокрема смартфонів і планшетів, що функціонують на операційній системі Android, забезпечуючи зручний доступ до управління фінансами в будь-який час і в будь-якому місці.

Підстави для розробки

Підставою для розробки даного програмного продукту є методичні рекомендації та нормативні документи, затверджені Волинським національний університетом імені Лесі Українки, факультетом інформаційних технологій і математики. Тема розробки має позначення – «Розробка мультикористувацької мобільної системи бюджетного контролю з функціями звітності за допомогою фреймворку Spring».

Призначення розробки

Функціональне призначення програмного продукту полягає в забезпеченні зручного інструменту для контролю бюджетів, що дозволяє користувачам вести облік доходів і витрат, управляти спільними бюджетами та генерувати детальні фінансові звіти. Програма надає можливість реєстрації та автентифікації користувачів, організації мультикористувацького доступу.

Експлуатаційне призначення системи визначається її здатністю працювати на мобільних пристроях під керуванням Android та в інтегрованому середовищі

серверної архітектури на базі Spring Boot і PostgreSQL. Програмний продукт розрахований на стабільну роботу в режимі реального часу, забезпечення високого рівня безпеки даних та зручного користувацького інтерфейсу, що дозволяє ефективно керувати фінансовими ресурсами як для окремих користувачів, так і для групових застосувань.

Вимоги до програмного продукту

Вимоги до функціональних характеристик. Програмний продукт повинен забезпечувати реалізацію низки основних функцій, які складають його функціональне наповнення. Зокрема, система має надавати можливості реєстрації та автентифікації користувачів із захищеним зберіганням даних, управління особистими та спільними фінансовими записами, додавання, редагування та видалення операцій, а також генерації детальних фінансових звітів із візуалізацією у вигляді таблиць, діаграм та графіків.

Організація вхідних даних передбачає можливість введення числових значень (суми операцій), дат, вибір категорій витрат та доходів, текстових описів операцій і додаткових приміток. Ці дані повинні оброблятися в режимі реального часу та зберігатися в базі даних з дотриманням принципів цілісності та безпеки. Вихідні дані формуються у вигляді звітів, які повинні бути зрозумілими, структурованими та доступними для подальшого аналізу користувачами.

Часові характеристики роботи системи визначають максимальний час відповіді для ключових операцій: наприклад, час обробки запиту на реєстрацію не повинен перевищувати 3 секунд, а генерація звітів – 5 секунд при стандартному навантаженні. Програмний продукт має забезпечувати стабільну паралельну обробку запитів від багатьох користувачів, що гарантує ефективну роботу системи навіть при високій інтенсивності операцій.

Умови експлуатації. Програмний продукт розроблено для роботи в стандартних умовах експлуатації інформаційних систем, де забезпечено стабільне електроживлення, контроль температурного режиму та захист від механічних пошкоджень як для серверного обладнання, так і для мобільних пристроїв. Система повинна працювати на мобільних пристроях з операційною

системою Android (версія не нижче 8.0) та у серверному середовищі з підтримкою Spring Boot і PostgreSQL, що гарантує відповідність заданим характеристикам при нормальному рівні мережевого з'єднання та обчислювальних ресурсів.

Обслуговування системи включає регулярне технічне оновлення програмного забезпечення, моніторинг працездатності серверів, резервне копіювання даних та своєчасне усунення виявлених збоїв. Для забезпечення безперервної роботи та оперативного реагування на можливі неполадки необхідно залучення кваліфікованого персоналу, який має досвід роботи з технологіями Spring Boot, PostgreSQL, Android Studio та Kotlin. Рекомендована кількість спеціалістів для первинного етапу впровадження – щонайменше 2-3 особи, здатні здійснювати як технічну підтримку, так і планове обслуговування системи.

Вимоги до складу і параметрів технічних засобів. Програмний продукт охоплює серверну частину, мобільний додаток та базу даних, тому для його ефективної роботи необхідно забезпечити відповідний склад технічних засобів із заданими характеристиками.

Для серверної частини рекомендується використовувати сервер з процесором не менше двоядерного (наприклад, Intel Xeon або AMD еквівалент) з тактовою частотою не менше 2.0 ГГц, оперативною пам'яттю – мінімум 8 ГБ, SSD-накопичувачем об'ємом від 50 ГБ та мережевим адаптером зі швидкістю передачі даних не менше 100 Мбіт/с.

Для бази даних, що працює на PostgreSQL, необхідно забезпечити стабільну роботу при високому навантаженні, що передбачає використання серверного обладнання з можливістю масштабування ресурсів, регулярного резервного копіювання та моніторингу продуктивності.

Мобільний додаток орієнтовано на пристрої з операційною системою Android версії 8.0 або вище, з мінімальною оперативною пам'яттю 2 ГБ, дисплеєм з роздільною здатністю не нижче 1280×720, підтримкою стабільного інтернет-з'єднанням для забезпечення безперебійного доступу до сервісу.

Вимоги до маркування і упаковки. Програмний виріб повинен містити чітке маркування, яке ідентифікує назву продукту, номер версії, ліцензійні умови, реквізити розробника та інші необхідні відомості. Маркування має бути розміщене як на інтерфейсі програмного забезпечення, так і на всіх супровідних документах і носіях інформації. У випадку фізичного розповсюдження програмного продукту упаковка повинна забезпечувати захист програмного забезпечення від пошкоджень під час транспортування та зберігання. Упаковка включає супровідну документацію, інструкції з встановлення і використання, а також контактні дані служби підтримки. При електронному розповсюдженні застосунку використовується формат завантажуваного архіву, який містить як програмний код, так і електронну документацію. Варіанти і способи упаковки повинні гарантувати зручність розпізнавання та ідентифікації продукту, а також забезпечувати його безпечну доставку до кінцевого користувача.

Вимоги до транспортування і збереження. У разі фізичного транспортування, програмний продукт повинен бути упакований у надійні матеріали, які забезпечують захист від механічних пошкоджень і впливу вологи чи пилу. Програмне забезпечення, що передається через електронні канали (наприклад, через інтернет), повинно бути зашифроване і перевірене на наявність вірусів чи шкідливих програм.

Місце збереження програмного продукту повинно бути відповідним для безпечного зберігання даних на серверних платформах або в хмарних середовищах, забезпечуючи резервне копіювання та захист від несанкціонованого доступу. Програмне забезпечення має зберігатися в середовищах з регулярними оновленнями безпеки, захищених від потенційних загроз. Крім того, необхідно дотримуватись вимог щодо доступу до програмного продукту на всіх етапах транспортування та зберігання, що включає шифрування даних і контроль доступу до сервера з обмеженням доступу для неавторизованих осіб.

Вимоги до програмної документації

Програмна документація повинна включати повний комплект документів, необхідних для ефективної розробки, експлуатації та підтримки програмного продукту. Основний склад програмної документації включає такі розділи:

1. технічне завдання (ТЗ), яке містить опис функціональних вимог до програмного продукту, а також технічні характеристики, що визначають параметри програмного засобу.

2. документація для розробників, яка повинна включати опис архітектури програмного забезпечення, структури даних, алгоритмів і методів, що використовуються, а також інструкції щодо налаштування та розгортання серверного і мобільного компонентів системи.

3. користувацька документація, що містить інструкції з використання програми, опис функціональних можливостей для кінцевого користувача, а також рекомендації щодо налаштування і вирішення основних проблем.

4. документація для тестування, яка має включати плани тестування, сценарії, звіти про проведені тести та результати перевірок системи на відповідність вимогам.

5. інструкції з безпеки, що описують методи захисту даних та безпеку програмного продукту під час експлуатації, а також процедури відновлення після аварійних ситуацій або збоїв.

За необхідності можуть бути додатково розроблені спеціальні вимоги до документації, які стосуються таких аспектів, як локалізація програмного продукту, адаптація до особливостей певних регіонів або вимоги до відповідності міжнародним стандартам безпеки інформації. Всі документи повинні бути чіткими, структурованими та зрозумілими для цільової аудиторії, а також оновлюватися відповідно до змін у програмному продукті.

Техніко-економічні показники

Орієнтовна економічна ефективність розробки системи бюджетного контролю полягає в скороченні витрат на управління фінансами завдяки автоматизації процесів, зниженню помилок і підвищенню точності звітів.

Система дозволить зменшити витрати на адміністративні функції і скоротити час обробки даних.

Передбачувана річна потреба в продукті розраховується за кількістю потенційних користувачів серед малого і середнього бізнесу, а також серед приватних осіб.

Економічні переваги цієї розробки в порівнянні з аналогами полягають у зручності використання, гнучкості інтеграції та покращеній безпеці даних. Вартість розробки нижча за існуючі рішення, що забезпечує високу економічну ефективність.

Стадії і етапи розробки

Першою стадією є ***аналіз вимог***, де проводиться збір та аналіз вимог до системи, визначення основних функціональних та нефункціональних характеристик. У результаті цієї стадії розробляється технічне завдання та документація щодо архітектури системи. Термін виконання цієї стадії складає 3-5 днів, а виконавцями будуть аналітик і архітектор системи.

Наступною є ***стадія проєктування***, під час якої здійснюється створення проекту архітектури мобільного додатку, серверної частини на базі Spring, бази даних PostgreSQL та інтеграції з іншими системами. Також оформлюються технічні рішення та проектування бази даних. Термін виконання цієї стадії – 1 тиждень, а виконавцями є розробники та архітектор.

На етапі ***розробки програмного продукту*** реалізується мобільний додаток в Android Studio, бекенд на Spring, налаштовується база даних та інтеграція системи. Цей етап займає 2 тижні, і в ньому беруть участь розробники.

Після цього проводиться ***тестування та налагодження***, що включає перевірку функціональних можливостей, виправлення помилок, тестування безпеки та навантаження, а також інтеграції. Тестування триватиме до 3 днів, а виконавцями є тестувальники та розробники.

Завершальним етапом є ***впровадження та документообіг***, що передбачає підготовку готового рішення до виробничого середовища, підготовку та затвердження експлуатаційної документації та інструкцій для користувачів. Цей

етап займає декілька днів, а виконавцями є менеджер проєкту, аналітик та розробники.

Останнім етапом є ***підтримка та обслуговування***, що здійснюється після впровадження продукту та включає усунення помилок, оновлення та оптимізацію системи, і триває на постійній основі. Виконавцями цієї стадії є технічна підтримка.

На кожному етапі необхідно узгоджувати та затверджувати відповідні документи, зокрема технічне завдання, проектну документацію, інструкції для користувачів, а також звіти про виконану роботу.

Порядок контролю і приймання

Перш за все, для контролю якості і працездатності системи будуть проведені кілька видів випробувань. Зокрема, це функціональне тестування, яке забезпечить перевірку всіх основних функцій системи на відповідність вимогам технічного завдання. Буде здійснено тестування безпеки для перевірки захисту даних користувачів, а також навантажувальне тестування, яке дасть змогу оцінити стабільність і ефективність роботи системи під високими навантаженнями.

Крім того, будуть проведені інтеграційні тести, щоб перевірити взаємодію між різними частинами системи (мобільний додаток, сервер, база даних). Всі ці випробування допоможуть переконатися в безперебійному функціонуванні системи та відповідності її вимогам.

Приймання роботи відбуватиметься після успішного проходження всіх етапів тестування та усунення виявлених помилок. Програмний продукт буде прийнятий, якщо він відповідає вимогам, зазначеним у технічному завданні, і забезпечує стабільну роботу без критичних помилок. За результатами приймання буде складено акт, який підтверджує завершення робіт і готовність продукту до впровадження.

Додаток Б.

Інструкція користувачу

Загальні відомості

Найменування програми: «Система бюджетного контролю «Budget Tracker»». Автор програми: Омелянчук Володимир Володимирович.

Програмний продукт «Budget Tracker» призначений для спрощеного обліку особистих та сімейних фінансів. Додаток складається з серверної частини, що працює під керуванням Spring Boot і зберігає дані в реляційній базі PostgreSQL, та мобільного Android-клієнта, що надає зручний інтерфейс для користувача. Інструкція описує основні можливості програми, вимоги до середовища та порядок взаємодії з системою.

Функціональне призначення

Додаток дозволяє реєструватися у системі та вести облік доходів і витрат із прив’язкою до обраних категорій. Користувач може створювати довільні категорії, додавати транзакції з сумами і нотатками, а також переглядати статистику за вибраними періодами. Крім того, передбачена можливість формування PDF-звіту з підсумками витрат і доходів, який можна вивантажити на пристрій або отримати електронною поштою. Система підтримує багатокористувацький режим і гарантує розмежування прав доступу за допомогою токенів JWT

Умови застосування програми

Для запуску серверної частини на локальному комп’ютері необхідно встановити Java 11+, PostgreSQL 13+ та виконати початкове налаштування бази даних відповідно до розділу впровадження. Сервіс запускається командою gradlew bootRun і доступний за адресою http://localhost:8080. Мобільний додаток сумісний з пристроями під керуванням Android 8.0 і вище. Перед початком роботи переконайтеся, що в константах клієнта вказано актуальну IP-адресу сервера та налаштовано мережевий доступ. Стабільне інтернет-підключення не

є обов’язковим, адже зміни фіксуються локально й синхронізуються автоматично при відновленні зв’язку

Повідомлення оператору

У разі неможливості виконати вхід система виводить повідомлення «Недійсні облікові дані» або «Помилка мережі. Неможливо підключитися до сервера. Перевірте ваше Інтернет з’єднання», після чого рекомендується перевірити правильність введених логіну й пароля, а також доступність сервера.

Опис роботи програми

Після встановлення і запуску клієнтського додатка користувач потрапляє на екран входу або реєстрації. Для створення нового облікового запису достатньо ввести email, пароль і підтвердити електронну пошту через надісланий лист. Після успішної автентифікації відкривається головний екран, де відображаються підсумкові значення доходів і витрат за поточний місяць. Перехід до списку транзакцій здійснюється через меню навігації. Щоб додати новий запис, натисніть на кнопку «+», заповніть поля суми, дати та категорії й підтвердіть операцію.

Для формування звіту перейдіть до розділу «Звіти», вкажіть період та натисніть «Генерувати». Після обробки на сервері PDF-файл підтягується у вигляді Base64-рядка та завантажується у файли програми. Вихід із програми виконується через кнопку «Вийти» в меню профілю.

Дотримуючись наведених інструкцій, користувач забезпечить безперебійну роботу програми, зможе своєчасно фіксувати всі фінансові операції та отримувати детальні звіти для аналізу свого бюджету.

АНОТАЦІЯ

Омелянчук В. В. – Розробка мультикористувацької мобільної системи бюджетного контролю з функціями звітності за допомогою фреймворку Spring. Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

Робота присвячена створенню мобільної системи, яка дозволяє кільком користувачам разом керувати бюджетом, контролювати витрати та доходи, а також формувати фінансові звіти. У проєкті використано фреймворк Spring для розробки серверної частини, а також реалізовано REST API для обміну даними з мобільним застосунком.

Під час виконання роботи було розглянуто різні варіанти побудови архітектури подібних систем, визначено ролі користувачів, способи доступу до даних і формування звітності. Створено функціонал реєстрації, авторизації, обліку фінансових операцій та спільного користування бюджетною інформацією.

Мета роботи – створити зручний, безпечний і ефективний мобільний застосунок для контролю бюджету в багатокористувацькому середовищі.

У результаті створено повноцінну систему, яка складається з серверної частини на базі Spring і клієнтської мобільної частини на Kotlin, що працює через API. Проведено тестування та надано рекомендації щодо подальшого покращення.

Ключові слова: мобільний застосунок, бюджет, Spring, багатокористувацька система, REST API, звітність, фінансові операції, авторизація, витрати, Java.