ЗМІСТ

[Перелік познак та скорочень 2](#_Toc197259663)

[Вступ 3](#_Toc197259664)

[1 Аналіз поточного стану та напрямків вдосконалення управління особистими фінансами 5](#_Toc197259665)

[1.1 Аналіз існуючих процесів управління особистими фінансами 5](#_Toc197259666)

[1.2 Моделювання та аналіз бізнес-процесів організації особистих фінансів 8](#_Toc197259667)

[1.3 Розробка специфікації вимог до програмного забезпечення 14](#_Toc197259668)

[1.3.1 Загальна концепція веб-застосунку обліку фінансів 14](#_Toc197259669)

[1.3.2 Загальний опис програмного забезпечення, що створюється 16](#_Toc197259670)

[1.3.3 Функціональні вимоги до розроблюваного програмного забезпечення 19](#_Toc197259671)

[1.3.4 Нефункціональні вимоги до розроблюваного програмного забезпечення 22](#_Toc197259672)

[2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ "ФІНАНСОВИЙ ПОМІЧНИК" 27](#_Toc197259673)

[2.1 Розробка діаграм поведінки веб-застосунку "Фінансовий Помічник" 27](#_Toc197259674)

[2.1.1 Розробка діаграм варіантів використання для різних категорій користувачів 27](#_Toc197259675)

[2.1.2 Уточнення реалізації варіантів використання 30](#_Toc197259676)

[2.2 Розробка бази даних для зберігання інформації 32](#_Toc197259677)

[2.2.1 Визначення бізнес-правил предметної області 32](#_Toc197259678)

[2.2.2 Розробка структури бази даних 34](#_Toc197259679)

[2.3 Розробка діаграм структури програмного забезпечення 43](#_Toc197259680)

# Перелік познак та скорочень

BPMN – Business Process Model and Notation;

CPU – Central Processing Unit;

CSV – Comma-Separated Values;

MTBF – Mean Time Between Failures;

MVP – Minimum Viable Product;

OS – Operating System;

QA – Quality Assurance;

RAM – Random Access Memory;

SDD – Software Design Description;

SRS – Software Requirements Specification;

UAT – User Acceptance Testing;

UI/UX – User Interface;

UML – Unified Modeling Language;

ПЗ – Програмне забезпечення;

СКБД – Система керування базами даних.

# Вступ

Сучасний ринок фінансових послуг та особистого бюджетування стикається зі зростанням потреби в інструментах, які б допомагали користувачам ефективно керувати своїми доходами та витратами. З одного боку, стрімка цифровізація фінансових операцій спрощує доступ до грошей, а з іншого — створює ризики неконтрольованих витрат, заборгованостей та втрати фінансової стабільності. Особистий фінансовий менеджмент стає критично важливим для досягнення довгострокових цілей, таких як накопичення, інвестування або погашення кредитів. Однак існуючі рішення часто не забезпечують достатньої гнучкості, автоматизації або захисту даних, що робить їх недоступними або незручними для широкого кола користувачів. Актуальність розробки програмного забезпечення для впорядкування особистих фінансів полягає в потребі поєднання простоти використання з потужними аналітичними інструментами, що дозволить користувачам не лише фіксувати транзакції, а й прогнозувати та оптимізувати свої фінансові потоки.

Предметна область дослідження стосується ключових аспектів управління особистими фінансами, таких як облік доходів і витрат, категоризація транзакцій, бюджетування та аналіз фінансових даних за допомогою візуалізації.

Метою дослідження є підвищення ефективності управління особистими фінансами для фінансових новачків. Досягнення цієї мети передбачається шляхом проектування програмного продукту, який надасть їм інструменти для повного контролю над фінансовими потоками, превентивного планування бюджету та інтерактивної візуалізації даних. Очікується, що використання такого рішення матиме позитивний вплив: економічний ефект полягатиме у зниженні ризиків фінансових втрат через несвоєчасні платежі або перевитрати, а соціальний – у підвищенні фінансової грамотності та формуванні фінансової дисципліни серед користувачів.

Об’єктом дослідження є процеси управління особистими фінансами, включаючи збір і обробку транзакцій та аналіз узагальнених даних про прибутки та витрати. Предметом дослідження виступають методи та підходи, необхідні для реалізації програмного продукту, зокрема: методи інтеграції з фінансовими сервісами (або імпорту даних), алгоритми автоматичної категоризації транзакцій, механізми інтерактивної візуалізації фінансових даних та інструменти забезпечення захисту інформації.

Реалізація поставленої мети включає виконання наступних завдань: проектування архітектури системи, реалізацію функціоналу для імпорту транзакцій, розробку інтерфейсу з інтерактивними звітами та забезпечення відповідності стандартам кібербезпеки. У сукупності це дозволить створити обґрунтування для продукту, який перетворює фінансове планування зі стресового обов’язку на інструмент для досягнення фінансової свободи.

# 1 Аналіз поточного стану та напрямків вдосконалення управління особистими фінансами

## 1.1 Аналіз існуючих процесів управління особистими фінансами

Підвищення рівня фінансової грамотності населення є важливим завданням, що відображено у результатах відповідних досліджень [1] та національних стратегіях розвитку [2]. Управління особистими фінансами є ключовим елементом фінансового добробуту. Під таким управлінням пропонується розуміти процеси обліку доходів та витрат, планування бюджету, аналіз фінансового стану та досягнення фінансових цілей [3]. Для однозначного розуміння термінів, що використовуються в контексті управління особистими фінансами, пропонується глосарій, наведений далі.  
1 Активи – все, що має цінність і належить особі (готівка, кошти на рахунках, нерухомість, інвестиції тощо).  
2 Бюджет – детальний план очікуваних доходів та запланованих витрат на певний період (наприклад, місяць або рік).  
3 Витрати – сума грошей, витрачена на товари, послуги або інші цілі.  
4 Доходи – усі грошові надходження особи за певний період (зарплата, відсотки, дохід від оренди тощо).  
5 Категоризація транзакцій – процес присвоєння кожній фінансовій операції (транзакції) певної категорії (наприклад, "Їжа", "Транспорт", "Комунальні послуги", "Розваги") для подальшого аналізу структури витрат.  
6 Особистий фінансовий менеджмент – процес управління особистими грошовими потоками через планування бюджету, облік доходів і витрат, заощадження, інвестування та управління боргами з метою досягнення фінансових цілей.  
7 Пасиви – фінансові зобов'язання особи (кредити, борги).  
8 Транзакція – окрема фінансова операція, що фіксує надходження (дохід) або вибуття (витрата) грошових коштів.  
9 Фінансова мета – конкретна ціль, пов'язана з грошима, яку особа прагне досягти (наприклад, накопичення на автомобіль, погашення кредиту, створення пенсійного фонду).  
10 Фінансовий звіт – документ або візуальне представлення, що підсумовує фінансовий стан особи за певний період (наприклад, звіт про доходи та витрати, балансовий звіт).

Визначення термінології дозволяє деталізувати зміст процесу управління фінансами. Для конкретизації аналізу варто зосередитись на типовому представнику цільової аудиторії: фінансовому новачку, наприклад, молодому фахівці віком 25-35 років, який отримує регулярний дохід, користується банківськими картками та онлайн-сервісами, але стикається з труднощами в систематичному обліку витрат, плануванні заощаджень та контролі над імпульсивними покупками [4].

На основі аналізу типових практик управління фінансами для визначеної цільової аудиторії [5, 6], можна окреслити послідовність дій такої особи та пов'язані з ними проблеми. Процес зазвичай починається з отримання доходу, який надходить на банківські рахунки, можливо з кількох джерел, при цьому облік загальної суми доходу часто ведеться лише приблизно. Наступним кроком є щоденне здійснення витрат за допомогою карток, готівки та онлайн-платежів; тут виникає проблема відстеження дрібних витрат, що призводить до ситуації "незрозуміло куди зниклих" грошей наприкінці місяця.

Незважаючи на критичну важливість фінансового обліку, спроби його систематичного ведення фінансовими новачками часто наштовхуються на значні перешкоди та виявляються неефективними. Наприклад, хоча вони можуть переглядати банківські виписки, вони отримують лише сирий потік транзакцій без зручних інструментів для їх аналізу. Відсутність автоматичної категоризації та візуалізації не дозволяє їм швидко зрозуміти структуру витрат, виявити проблемні зони чи оцінити прогрес у досягненні фінансових цілей, перетворюючи виписку на малоінформативний список цифр.

Інший поширений підхід – використання електронних таблиць, таких як Microsoft Excel, – хоч і надає більше гнучкості, проте вимагає від людини не лише високої самодисципліни, але й значних витрат часу на рутинне ручне введення кожної транзакції та створення складних формул для аналізу. Це не тільки підвищує ризик помилок, які можуть призвести до невірних фінансових висновків, але й часто стає непосильним тягарем для зайнятих людей, через що ведення таблиць часто закидається, залишаючи їх без будь-якого фінансового контролю.

Здавалося б, виходом можуть стати прості мобільні додатки, проте вони також мають суттєві вади. Часто їх функціонал є надто обмеженим і не задовольняє потреби фінансових новачків, які прагнуть не лише фіксувати витрати, але й комплексно планувати бюджети та ставити фінансові цілі. Крім того, досвід їх використання може псуватися нав'язливою рекламою, а питання безпеки конфіденційних фінансових даних залишається відкритим. Важливою проблемою, особливо для українського ринку, є відсутність або обмежена підтримка інтеграції з місцевими банками, що зводить нанівець зручність автоматичного збору даних. Як наслідок, жоден з цих поширених методів не забезпечує поєднання зручності, автоматизації, надійності та аналітичних можливостей, необхідних для структурного та регулярного керування особистими фінансами в сучасних умовах [7].

Внаслідок відсутності систематизованих даних, аналіз фінансового стану, такий як розуміння структури витрат чи реального залишку коштів, стає ускладненим. Планування бюджету та постановка фінансових цілей (накопичення, велика покупка) часто носять інтуїтивний характер, без чіткої стратегії. Це призводить до відсутності ясного розуміння власної фінансової ситуації, ризику перевитрат, труднощів із накопиченням та фінансового стресу. Таким чином, основні недоліки існуючих підходів полягають у відсутності автоматизації збору даних, складності категоризації та аналізу, а також браку інструментів для зручного планування та візуалізації фінансової картини [5].

## 1.2 Моделювання та аналіз бізнес-процесів організації особистих фінансів

Аналіз предметної області, проведений у попередньому пункті, виявив низку недоліків у типових підходах до управління особистими фінансами цільовою аудиторією – молодими фахівцями. Ці недоліки, пов'язані переважно з ручним обліком, відсутністю систематизації та аналітичних інструментів, вказують на необхідність вдосконалення існуючих процесів шляхом впровадження спеціалізованого програмного забезпечення. Для чіткого розуміння того, як саме пропоноване ПЗ змінить та покращить діяльність користувача, необхідно формалізувати та візуалізувати ці процеси. Моделювання бізнес-процесів (БП) є ключовим інструментом для досягнення цієї мети. Згідно зі стандартом ISO 9000-2001, процес визначається як «набір взаємопов'язаних або взаємодіючих заходів, що перетворюють входи на виходи» [8]. Бізнес-процес, у більш широкому розумінні, це цілеспрямована послідовність дій, яка використовує ресурси для перетворення входів на цінні для споживача виходи [8]. Моделювання дозволяє документувати, аналізувати та оптимізувати ці послідовності дій.

У даному розділі буде застосовано нотацію BPMN (Business Process Model and Notation) версії 2.0 [9] для створення моделей двох основних бізнес-процесів, що підлягають автоматизації в рамках розробки системи управління особистими фінансами: процесу обліку доходів та витрат, а також процесу створення та контролю бюджету. Для кожного з них будуть розроблені дві моделі: модель "AS-IS" (Як є), що відображає поточний стан процесу без використання пропонованого ПЗ, та модель "TO-BE" (Як буде), що представляє вдосконалений процес за участю розроблюваного програмного забезпечення. Аналіз моделі "AS-IS" дозволить виявити конкретні слабкі місця, а порівняння її з моделлю "TO-BE" наочно продемонструє переваги впровадження розроблюваного ПЗ. Використано діаграми бізнес-процесів (Business Process Diagrams, BPD) в нотації BPMN.

Розглянуто перший ключовий процес – облік доходів та витрат. Його метою є систематична фіксація всіх фінансових транзакцій користувача за певний період для отримання достовірної картини руху коштів. Входами для цього процесу слугує інформація про транзакції (банківські виписки, SMS, чеки, пам'ять про готівкові операції) та визначені користувачем категорії витрат/доходів. Виходом є зареєстрований та категоризований список транзакцій, наприклад, у файлі Excel чи зошиті. Власником та основним виконавцем процесу в стані "AS-IS" є сам фінансовий новачок, який використовує такі ресурси, як комп'ютер з Excel, зошит, ручку, доступ до банківських сервісів та власний час. Модель цього процесу в стані "AS-IS" наведена на рисунку 1.1.

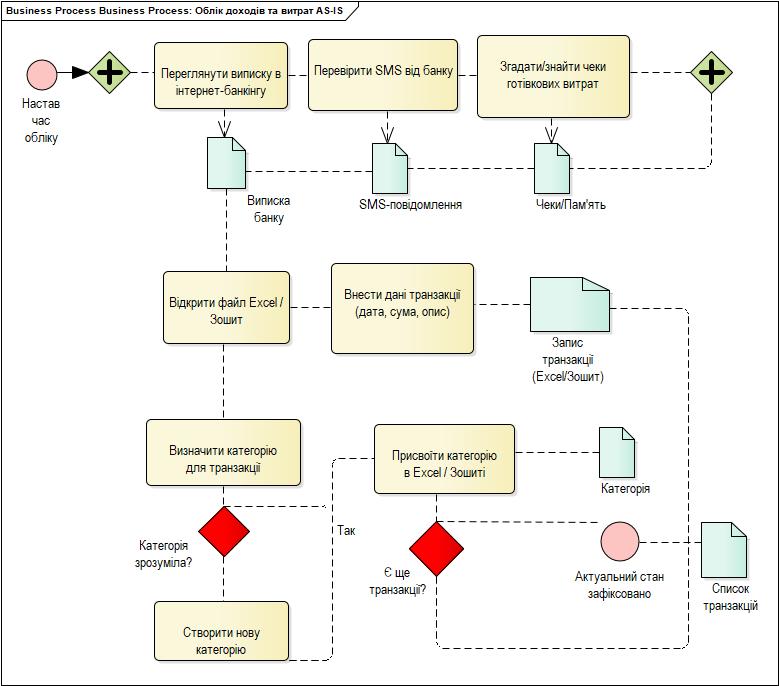


Рисунок 1.1 - Діаграма бізнес-процесу   
"Облік доходів та витрат" (AS-IS)

Аналіз цієї моделі "AS-IS" (рис. 1.1) чітко демонструє основні проблеми поточного процесу обліку. По-перше, це висока трудомісткість, оскільки фінансовий новачок витрачає значний час на ручний збір, введення та категоризацію даних. По-друге, процес схильний до помилок через ручне введення, що призводить до неточностей та пропусків транзакцій. По-третє, дані часто є несвоєчасними, бо облік ведеться із запізненням, не даючи актуальної картини. Також виникають проблеми з непослідовною категоризацією, коли категорії визначаються "на око", що ускладнює аналіз. Дані залишаються фрагментованими у різних джерелах, і сам процес є психологічно обтяжливим, що часто призводить до відмови від обліку взагалі. Ці слабкі місця роблять існуючий процес неефективним.

Тепер буде аналізуватись модель "TO-BE" для цього ж процесу, яка передбачає використання розроблюваного ПЗ. Модель "TO-BE" для процесу обліку доходів та витрат, що враховує використання ПЗ, зображена на рисунку 1.2.

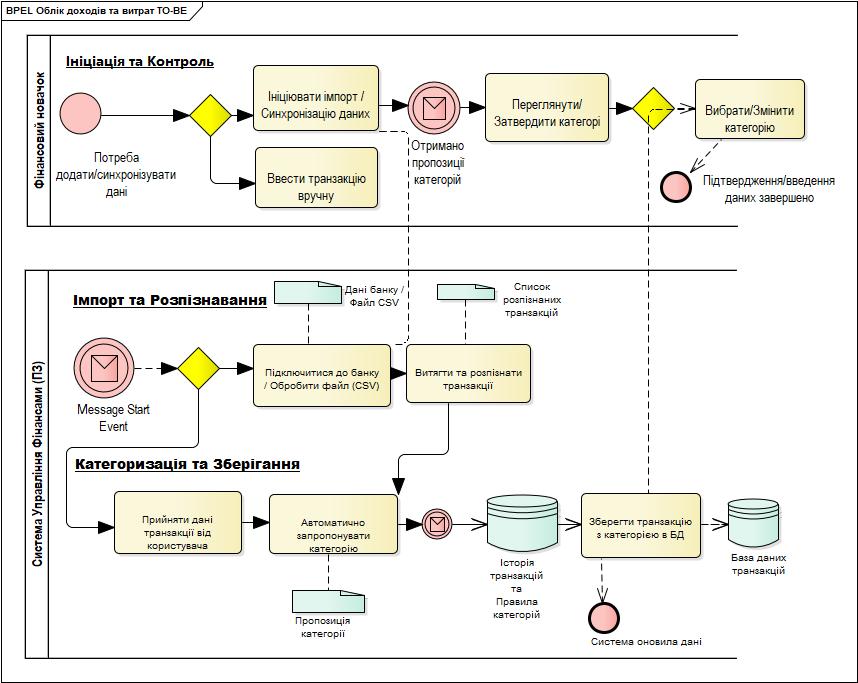


Рисунок 1.2 – Діаграма бізнес-процесу   
"Облік доходів та витрат" (TO-BE)

Модель "TO-BE" (рис. 1.2) демонструє суттєві вдосконалення. Завдяки автоматизації збору даних через імпорт або синхронізацію значно скорочується час користувача та підвищується точність. Дані стають актуальними та доступними швидко. Інтелектуальна категоризація прискорює процес та забезпечує його послідовність, залишаючи фінансовому новачку лише контроль. Усі дані централізуються в єдиній базі, що полегшує подальший аналіз. Процес стає значно простішим та менш обтяжливим, знижуючи психологічний бар'єр.

Другим важливим бізнес-процесом є створення та контроль бюджету. Його мета полягає у плануванні очікуваних доходів та витрат за категоріями на певний період (місяць, рік) та відстеженні фактичного виконання плану для контролю витрат та досягнення фінансових цілей. Входами є дані про доходи, список категорій, бажані ліміти та дані про фактичні транзакції. Виходами є бюджетний план, звіти про його виконання та сповіщення. В стані "AS-IS" власником та виконавцем є фінансовий новачок, який використовує комп'ютер з Excel, зошит, записи про транзакції та власні аналітичні здібності. Графічне представлення процесу "AS-IS" для створення та контролю бюджету показано на рисунку 1.3.

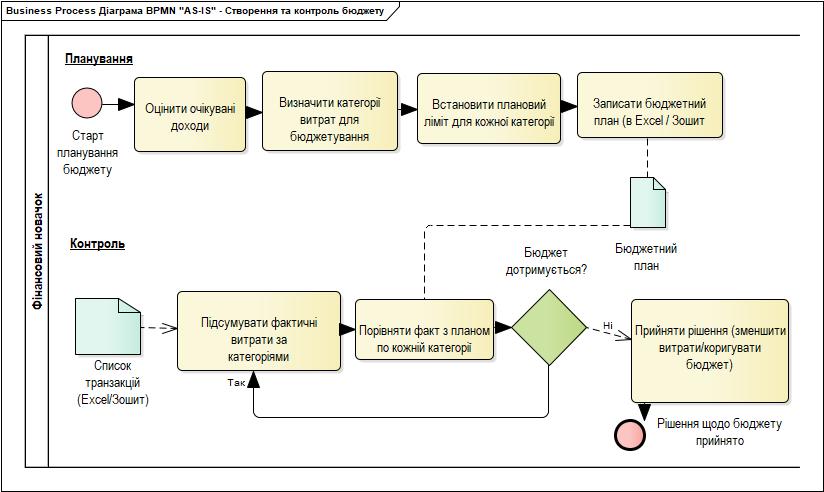


Рисунок 1.3 - Діаграма бізнес-процесу "Створення та контроль бюджету" (AS-IS)

Аналіз цієї моделі "AS-IS" (рис. 1.3) виявляє значні недоліки. Планування ускладнене через брак аналітичної бази для встановлення реалістичних лімітів. Контроль є трудомістким, оскільки вимагає ручного підсумовування та порівняння, що схильне до помилок. Фінансовий новачок отримує інформацію про виконання бюджету несвоєчасно, часто вже після виникнення перевитрат, що зумовлює реактивний підхід до управління. Відсутність наочної візуалізації ускладнює швидку оцінку ситуації, а сам процес є недостатньо гнучким для оперативних коригувань. Це робить бюджетування громіздким та неефективним.

Модель "TO-BE" для процесу створення та контролю бюджету показує, як ПЗ може вирішити ці проблеми. Модель "TO-BE" для процесу створення та контролю бюджету за допомогою розроблюваного ПЗ представлена на рисунку 1.4.

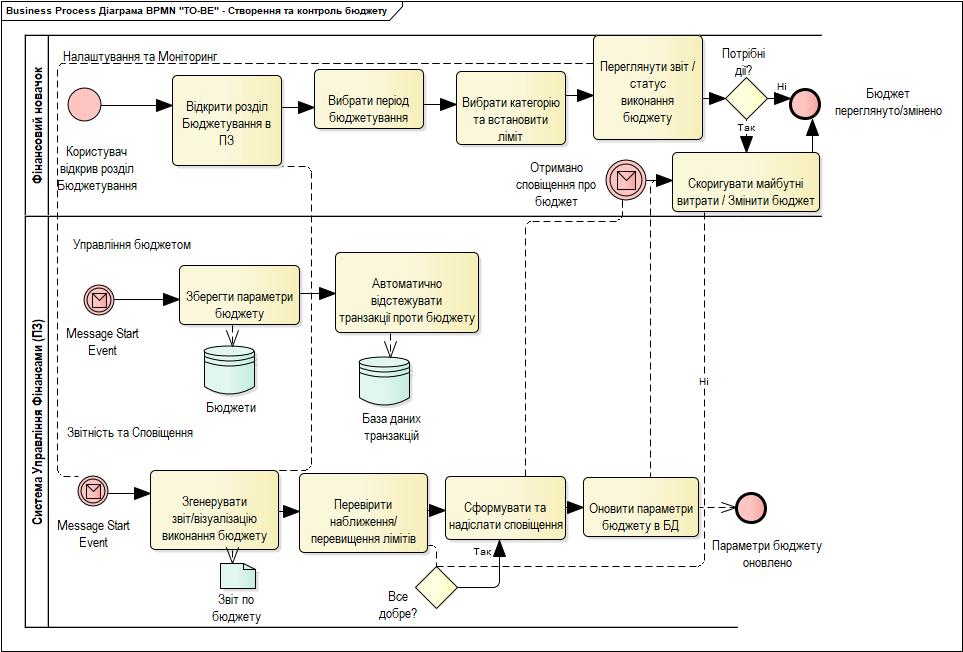


Рисунок 1.4 - Діаграма бізнес-процесу "Створення та контроль бюджету" (TO-BE)

Впровадження програмного забезпечення радикально змінює процес бюджетування, що відображено в моделі "TO-BE" (рис. 1.4). Планування спрощується завдяки можливим підказкам системи на основі історичних даних. Контроль стає автоматизованим, оскільки система сама зіставляє транзакції з бюджетом. Фінансовий новачок отримує інформацію в реальному часі та може швидко оцінити стан виконання бюджету завдяки наочній візуалізації. Система забезпечує проактивність через сповіщення, що дозволяє запобігати перевитратам. Процес стає гнучкішим, дозволяючи легко коригувати бюджет.

Отже, моделювання бізнес-процесів "Облік доходів та витрат" та "Створення та контроль бюджету" за допомогою нотації BPMN [9] дозволило чітко візуалізувати поточний стан ("AS-IS", рис. 1.1, 1.3) та запропонований майбутній стан ("TO-BE", рис. 1.2, 1.4) цих ключових аспектів управління особистими фінансами. Аналіз моделей "AS-IS" підтвердив неефективність, трудомісткість та схильність до помилок ручних методів [8]. Виявлені слабкі місця, такі як несвоєчасність даних, непослідовність категоризації та брак проактивності, безпосередньо перешкоджають досягненню фінансової дисципліни та цілей фінансового новачка. Моделі "TO-BE" наочно демонструють, як розроблюване програмне забезпечення інтегрується в діяльність користувача та трансформує її. Завдяки автоматизації, інтелектуальним функціям, звітності в реальному часі та проактивним сповіщенням, пропоноване ПЗ усуває ключові недоліки моделей "AS-IS". Система бере на себе рутинні завдання, залишаючи користувачеві контроль та прийняття стратегічних рішень [5, 6]. Таким чином, моделювання підтверджує доцільність розробки програмного забезпечення та слугує основою для подальшої деталізації вимог до системи [10, 11], забезпечуючи відповідність розробки реальним потребам цільової аудиторії [4].

## 1.3 Розробка специфікації вимог до програмного забезпечення

### 1.3.1 Загальна концепція веб-застосунку обліку фінансів

Актуальність розробки спеціалізованого програмного забезпечення (ПЗ) в цій сфері зумовлена зростаючою складністю фінансових інструментів та потребою користувачів у зручних та ефективних засобах контролю власних коштів [4]. При розробці ПЗ для управління особистими фінансами необхідно враховувати низку чинників, які суттєво впливатимуть на вибір архітектури, функціоналу та технологій [10, 11].

Важливими є особливості самої предметної області. Зокрема, фінансова інформація є вкрай конфіденційною, що висуває високі вимоги до стандартів безпеки та захисту даних. Система повинна забезпечувати точність обліку всіх транзакцій. Для легкого сприйняття фінансової інформації користувачами необхідна наочна візуалізація у вигляді звітів та графіків (приклад наведено на рисунку 1.5). Користувачі також зацікавлені в автоматизації, особливо в мінімізації ручного введення даних, наприклад, через імпорт файлів або (в перспективі) інтеграцію з банками. Крім того, слід враховувати різноманітність потреб користувачів, які можуть мати різні рівні фінансової грамотності та різні цілі (від простого обліку до детального бюджетування та інвестування).

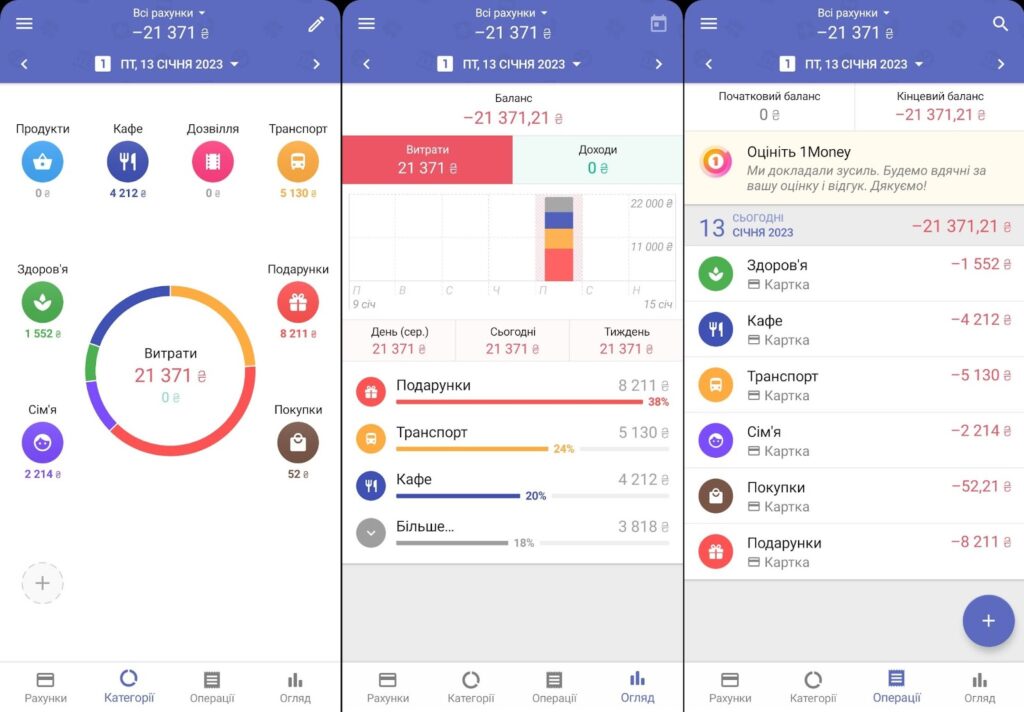


Рисунок 1.5 – Приклад якісної візуалізації у мобільному додатку

Значний вплив мають характеристики цільової аудиторії, визначеної як молоді фахівці. Вони є технічно обізнаними користувачами, тому очікують сучасний та інтуїтивно зрозумілий інтерфейс (UI/UX). Через обмежений час їм потрібен швидкий та зручний інструмент. Також вони орієнтовані на досягнення цілей, тому зацікавлені не лише в обліку, а й у функціях, що допомагають у цьому.

Аналіз існуючих рішень також є важливим чинником. Конкурентне середовище вимагає від нового ПЗ певних переваг, а неефективність "успадкованих" підходів (ручний облік) підтверджує потребу в автоматизованому рішенні.

Нарешті, необхідно враховувати проектні ресурси та обмеження, характерні для курсової роботи. Обмежений час розробки вимагає фокусування на ключовому функціоналі (MVP). Вибір технологій залежить від знань розробника. Відсутність бюджету та регуляторні складнощі обмежують можливість реалізації прямої банківської інтеграції, вимагаючи альтернатив (імпорт CSV).

Аналіз предметної області та існуючих процесів виявив значну потребу в автоматизованому, зручному та безпечному інструменті, оскільки традиційні методи є неефективними [7, 4]. Ключовими вимогами до майбутньої системи, що випливають з цього аналізу, є: простота обліку транзакцій, гнучка категоризація, потужні засоби аналізу та візуалізації, інструменти для бюджетування та відстеження цілей, а також надійний захист інформації. Врахування вищезазначених чинників впливу є основою для розробки програмного продукту, що відповідає реальним потребам користувачів в межах ресурсних обмежень. Загальна концепція такого проекту може бути представлена як на рисунку 1.6.

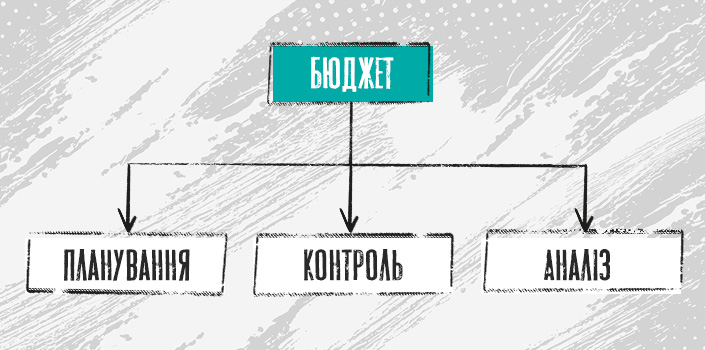


Рисунок 1.6 – Схема концепції проєкту

Таким чином, після проведення аналізу предметної області (п. 1.1) та моделювання ключових бізнес-процесів (п. 1.2), що продемонструвало необхідність та переваги впровадження спеціалізованого програмного забезпечення, наступним кроком є формалізація вимог до цього ПЗ. Розробка вимог є критично важливою діяльністю, що забезпечує узгоджене розуміння функціональності та характеристик майбутньої системи між усіма зацікавленими сторонами [10]. Згідно зі стандартом ISO/IEC/IEEE 29148 [10], результатом цього процесу є специфікація вимог до програмного забезпечення (SRS), яка слугує основою для подальших етапів розробки. У наступних підрозділах вимоги будуть представлені у форматі, адаптованому до структури курсової роботи, з використанням класифікацій вимог [11, 12] та дотриманням принципів якості формулювань [10].

### 1.3.2 Загальний опис програмного забезпечення, що створюється

Цей підрозділ надає загальний контекст для програмного продукту, вимоги до якого деталізуються далі.

Продуктом, що розроблюється, є веб-додаток під назвою "Фінансовий Помічник", призначений для організації та управління особистими фінансами. Система надасть користувачам інструменти для відстеження доходів і витрат, створення та моніторингу бюджетів, категоризації транзакцій, встановлення та відстеження фінансових цілей, а також аналізу фінансового стану за допомогою звітів та візуалізацій. Основне призначення ПЗ – допомогти користувачам (зокрема, цільовій аудиторії, визначеній у п. 1.1 – молодим фахівцям) отримати кращий контроль над своїми фінансами, підвищити фінансову дисципліну та ефективніше досягати своїх фінансових цілей. Перевагами системи є автоматизація рутинних операцій, таких як імпорт даних та їх категоризація, централізоване зберігання фінансової інформації, наочне її представлення та можливість отримання проактивних сповіщень. Програмне забезпечення буде функціонувати як веб-додаток, доступний через сучасні веб-браузери (Chrome, Firefox, Safari, Edge останніх версій) на настільних комп'ютерах та ноутбуках. Система працюватиме на стандартному серверному оточенні під керуванням ОС Linux та взаємодіятиме з системою керування базами даних PostgreSQL. Інших специфічних програмних компонентів для співіснування на стороні клієнта не передбачається, окрім стандартного веб-браузера. Розробка цього ПЗ безпосередньо пов'язана з бізнес-цілями, визначеними раніше, такими як підвищення фінансової грамотності користувачів та зниження ризиків фінансових втрат через невдале управління коштами.

Щодо перспективи програмного продукту, "Фінансовий Помічник" розробляється як новий, самостійний продукт. Він має на меті запропонувати сучасну та зручну альтернативу існуючим методам управління фінансами, таким як електронні таблиці, паперові записи або менш функціональні мобільні додатки. Система не є частиною більшого програмного комплексу чи сімейства продуктів. Вона призначена для безпосереднього використання кінцевими користувачами для управління їхніми особистими фінансами. Основні зовнішні взаємодії системи включають взаємодію з користувачем через веб-інтерфейс та потенційну взаємодію з файлами експорту банківських даних (напр., у форматі CSV) для імпорту транзакцій. Контекст системи та її основні взаємодії схематично зображені на рисунку 1.7.

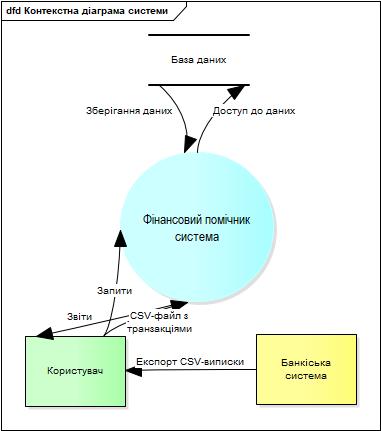


Рисунок 1.7 - Контекстна діаграма системи "Фінансовий Помічник"

Основним класом користувачів є "Індивідуальний користувач", що відповідає профілю молодого фахівця (25-35 років), визначеному в п. 1.1. Характеристики цього класу включають впевнене володіння комп'ютером та веб-технологіями, досвід використання онлайн-сервісів, а також рівень фінансової грамотності від початкового до середнього. Ці користувачі зазвичай мають обмежений час, цінують швидкість та ефективність, прагнуть до кращого контролю над фінансами та досягнення цілей. Очікується регулярне використання системи. Інших класів користувачів на даному етапі не передбачається, і вимоги до зручності використання орієнтовані саме на цей основний клас. Документація користувача буде інтегрована безпосередньо в продукт у вигляді вбудованої онлайн-довідки та контекстних підказок, без окремих файлів посібників для поточної версії.

Визначення основних термінів предметної області наведено у глосарії в пункті 1.1, а список скорочень та абревіатур – на початку пояснювальної записки.

### 1.3.3 Функціональні вимоги до розроблюваного програмного забезпечення

Цей розділ описує основні функції, які повинна виконувати система "Фінансовий Помічник", визначаючи, що саме система повинна робити [11]. Система повинна забезпечувати наступні основні функціональні можливості:

Управління транзакціями:

* Система повинна дозволяти фінансовому новачку вручну вводити доходи та витрати із зазначенням дати, суми, категорії, рахунку (готівка, картка) та опису.
* Система повинна підтримувати імпорт транзакцій з файлів формату CSV, надаючи можливість налаштування відповідності колонок.
* Система повинна надавати фінансовому новачку можливість редагувати та видаляти існуючі транзакції.
* Система повинна дозволяти фінансовому новачку вручну категоризувати транзакції, обираючи категорію зі списку, а також пропонувати категорію автоматично (на основі правил або історії попередніх категоризацій).

Управління бюджетами:

* Система повинна дозволяти фінансовому новачку створювати бюджети на певний період (місяць, рік) для обраних категорій витрат та встановлювати для них планові ліміти.
* Система повинна автоматично розраховувати та відстежувати фактичні витрати за бюджетними категоріями.
* Система повинна візуально відображати прогрес виконання бюджету (наприклад, за допомогою прогрес-барів або аналогічних елементів).
* Система повинна надавати можливість копіювати структуру існуючого бюджету для створення нового на наступний період.

Управління фінансовими цілями:

* Система повинна дозволяти фінансовому новачку створювати фінансові цілі із зазначенням назви, цільової суми та бажаної дати досягнення.
* Система повинна надавати можливість пов'язувати конкретні заощадження або рахунки з фінансовими цілями.
* Система повинна розраховувати та відображати прогрес досягнення фінансових цілей.

Звітність та аналітика:

* Система повинна генерувати звіт про доходи та витрати за обраний фінансовим новачком період, з можливістю фільтрації за категоріями та рахунками.
* Система повинна надавати візуалізацію структури витрат та доходів (наприклад, у вигляді кругових діаграм).
* Система повинна відображати динаміку витрат, доходів та залишку коштів з часом (наприклад, у вигляді лінійних графіків).
* Система повинна генерувати звіт про виконання бюджету, що містить планові та фактичні показники, а також розраховані відхилення.

Управління рахунками та категоріями:

* Система повинна дозволяти фінансовому новачку додавати, редагувати та видаляти рахунки (готівка, банківські картки тощо), вказуючи назву, початковий баланс та валюту (початково - UAH).
* Система повинна дозволяти фінансовому новачку додавати, редагувати та видаляти категорії доходів та витрат.
* Система повинна підтримувати можливість створення ієрархії підкатегорій (бажано).

Аутентифікація та авторизація:

* Система повинна забезпечувати можливість реєстрації нових користувачів.
* Система повинна забезпечувати безпечний вхід зареєстрованих користувачів у систему за допомогою логіна (email) та пароля.
* Система повинна гарантувати, що користувач має доступ лише до його власних фінансових даних.

Окрім функцій, безпосередньо пов'язаних з предметною областю, система повинна реалізовувати ряд важливих допоміжних можливостей:

* Журнал аудиту (Logging): Система повинна вести журнал основних дій користувача (вхід, вихід, критичні операції) та важливих системних подій (помилки).
* Локалізація (Localization): Інтерфейс користувача та системні повідомлення мають бути реалізовані українською мовою.
* Безпека (Security): Система повинна включати базові механізми захисту даних та протидії поширеним веб-атакам (деталі в нефункціональних вимогах).
* Допомога (Online help): Система повинна надавати контекстні підказки та доступ до розділу довідки всередині програми.

Загальну структуру функціональних вимог та їх взаємозв'язок можна представити у вигляді діаграми вимог (рисунок 1.8).

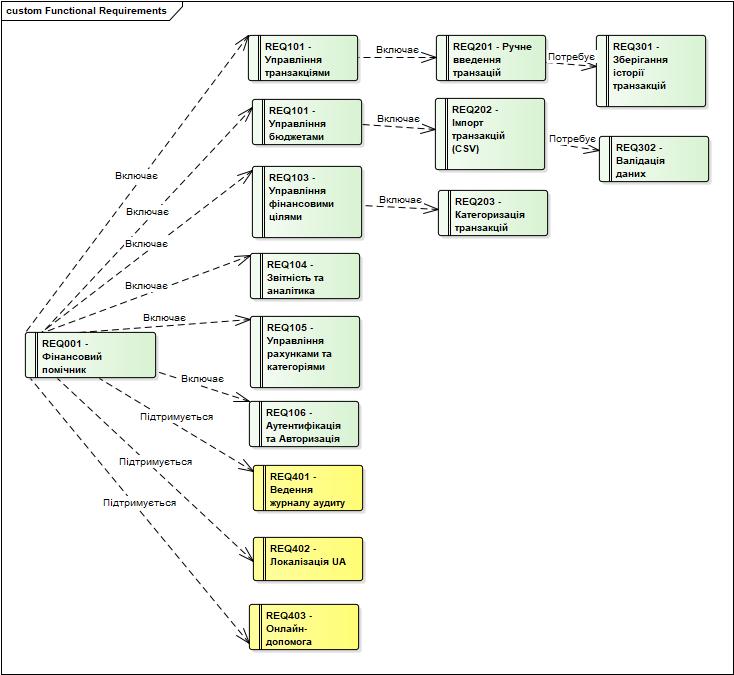


Рисунок 1.8 - Діаграма функціональних вимог системи "Фінансовий Помічник" (приклад у UML)

Система повинна також дотримуватись певних бізнес-правил, що випливають з логіки предметної області. Наприклад, сума транзакції при ручному введенні не може бути нульовою або від'ємною, дата транзакції за замовчуванням не може бути у майбутньому. Неможливо видалити категорію або рахунок, якщо вони використовуються в існуючих транзакціях, без попереднього перепризначення цих транзакцій. Система має попереджати користувача при спробі зберегти бюджет з явно нереалістичним співвідношенням доходів та витрат. Також система повинна надавати візуальні попередження або сповіщення при наближенні чи перевищенні бюджетних лімітів за категоріями.

### 1.3.4 Нефункціональні вимоги до розроблюваного програмного забезпечення

Нефункціональні вимоги визначають атрибути якості системи та обмеження, тобто *як* система повинна виконувати свої функції [11], і є критично важливими для користувацького досвіду та надійності продукту. Для структурування цих вимог використано категорії моделі FURPS+ [12] та стандарту [10].

Вимоги до продуктивності (Performance):

* Час відгуку інтерфейсу користувача на типові дії (відкриття сторінки, збереження транзакції, відображення списку) не повинен перевищувати 1 секунду при стабільному інтернет-з'єднанні та типовому навантаженні.
* Генерація стандартних звітів (про доходи/витрати, виконання бюджету за місяць) повинна займати не більше 3 секунд для обсягу даних до 5000 транзакцій.
* Імпорт файлу CSV, що містить до 200 транзакцій, повинен завершуватися протягом 5 секунд.
* Система повинна бути розрахована на стабільну роботу при одночасному доступі одного користувача (типовий сценарій використання).
* Система повинна ефективно працювати з обсягом даних користувача до 10 000 транзакцій та до 100 категорій/рахунків без суттєвої деградації продуктивності порівняно з роботою на менших обсягах.

Атрибути якості програмного забезпечення (Quality Attributes):

Зручність використання (Usability):

1. Інтерфейс повинен бути інтуїтивно зрозумілим для цільової аудиторії (визначеної в п. 1.1);
2. Новий користувач повинен мати змогу виконати базові сценарії (додати транзакцію, створити простий бюджет) протягом 10 хвилин ознайомлення з системою після реєстрації.
3. Доступ до найбільш частих операцій (додавання транзакції, перегляд поточного балансу/бюджету) повинен вимагати не більше 3-х кліків мишею з головної панелі (dashboard).
4. Система повинна надавати чіткі, зрозумілі та конструктивні повідомлення про помилки введення або системні проблеми.
5. Дизайн інтерфейсу повинен бути естетичним, сучасним та не перевантаженим елементами.

Надійність (Reliability):

* + Система повинна забезпечувати цілісність фінансових даних користувача; операції зміни даних повинні виконуватися транзакційно (на рівні СКБД).
  + Збій під час операції запису даних не повинен призводити до часткового збереження або пошкодження даних.

Надійність 2 (Reliability):

1) Оціночний середній час між збоями (MTBF), що призводять до недоступності критичних функцій, повинен бути не менше 100 годин роботи системи.

2) система повинна коректно обробляти некоректні або несподівані введення користувача (напр., спроба введення тексту в числове поле), видаючи відповідні повідомлення, а не призводячи до непередбачуваної поведінки чи аварійного завершення.

#### 1.3.4.5 Доступність (Availability)

Вимогою до доступності є така - очікувана доступність сервісу для користувачів повинна становити не менше 99.5% часу протягом місяця, за винятком планових вікон обслуговування (про які користувачі мають бути попереджені заздалегідь).

#### 1.3.4.6 Підтримуваність (Supportability / Maintainability)

До вимог цього напрямку слід віднести:

* вихідний код серверної та клієнтської частин повинен бути написаний з дотриманням загальноприйнятих стандартів кодування та найкращих практик для обраних технологій (Python/Django, JavaScript/React або Vue.js);
* код повинен бути структурованим, розділеним на логічні модулі/компоненти;
* код повинен бути достатньо коментованим для пояснення складних або неочевидних рішень;
* архітектура системи повинна бути модульною, щоб полегшити внесення змін та додавання нових функцій у майбутньому.

Портативність (Portability):

* Клієнтська частина (веб-інтерфейс) повинна коректно відображатися та функціонувати в останніх двох стабільних версіях основних веб-браузерів: Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge.
* Серверна частина програми повинна бути розгорнута за допомогою Docker-контейнерів, що забезпечує її легке перенесення між різними операційними середовищами (розробка, тестування, продуктивне середовище).

Масштабованість (Scalability):

* Архітектура системи повинна дозволяти вертикальне масштабування (збільшення ресурсів CPU, RAM, дискового простору сервера) для обробки зростаючого обсягу даних одного користувача без необхідності кардинальної зміни коду.
* Вимоги до безпеки та розмежування доступу (Security):

Аутентифікація**:**

* Система повинна вимагати від користувачів проходження аутентифікації за допомогою унікального логіна (адреса електронної пошти) та пароля.
* Паролі користувачів повинні зберігатися у базі даних виключно у хешованому вигляді з використанням криптографічно стійкої солі та сучасного алгоритму хешування (напр., bcrypt або Argon2).

Авторизація:

* Система повинна реалізовувати механізм розмежування доступу, що гарантує, що кожен аутентифікований користувач може отримувати доступ (читати, змінювати, видаляти) лише до власних фінансових даних.
* Будь-які спроби несанкціонованого доступу до даних інших користувачів повинні блокуватися та реєструватися в журналі безпеки.

Захист даних:

* Вся комунікація між клієнтським браузером та сервером повинна здійснюватися виключно через захищене з'єднання HTTPS.
* Будь-які потенційно чутливі дані конфігурації (напр., ключі до зовнішніх сервісів, якщо такі з'являться у майбутньому) повинні зберігатися у безпечний спосіб (напр., змінні середовища, зашифровані файли конфігурації).

Захист від веб-атак:

Система повинна мати вбудований захист від поширених веб-вразливостей згідно з рекомендаціями OWASP Top 10, включаючи, але не обмежуючись:

* Захист від SQL-ін'єкцій (через використання ORM або параметризованих запитів).
* Захист від міжсайтового скриптингу (XSS) (через належне екранування даних, що виводяться).
* Захист від підробки міжсайтових запитів (CSRF) (через використання CSRF-токенів для всіх запитів, що змінюють стан).

**Управління сесіями:**

* Система повинна використовувати безпечні, випадково генеровані ідентифікатори сесій.
* Сесійні куки повинні мати атрибути HttpOnly та Secure (при використанні HTTPS).
* Система повинна автоматично завершувати сесію користувача після визначеного періоду неактивності (напр., 30 хвилин).

#### 1.3.4 Обмеження проектування, розробки та впровадження

Існують певні фактори, що обмежують вибір розробника. Розробка має враховувати загальні принципи захисту персональних даних. Апаратні вимоги є помірними: клієнтська частина має працювати на типових комп'ютерах, а серверна – на стандартному VPS початкового рівня. Щодо інтерфейсів, ключовим є веб-інтерфейс користувача та підтримка імпорту лише з CSV-файлів; пряма банківська інтеграція не планується. Технологічний стек чітко визначений: Python/Django для бекенду, JavaScript з React (або Vue.js) для фронтенду, PostgreSQL як СКБД, Git для контролю версій та Docker для розгортання. Основна мова інтерфейсу – українська. Необхідно дотримуватись стандартів кодування для обраних технологій.

Також існують припущення та залежності. Припускається, що користувач має стабільний інтернет, можливість отримувати CSV-виписки та базові комп'ютерні навички. Проєкт залежить від стабільності зовнішніх бібліотек та фреймворків, інфраструктури хостингу та коректності даних у CSV-файлах користувача.

Ця специфікація вимог слугуватиме основою для наступних етапів розробки, фіксуючи узгоджене бачення продукту та критерії його успішної реалізації.

Попередній розділ був присвячений аналізу предметної області, моделюванню існуючих бізнес-процесів та формулюванню детальних вимог до програмного забезпечення "Фінансовий Помічник". Було визначено ключові функціональні та нефункціональні характеристики системи, а також обмеження, що впливають на її розробку. Цей розділ переходить від визначення "що" система повинна робити до проектування того, "як" вона буде це робити.

# 2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ "ФІНАНСОВИЙ ПОМІЧНИК"

Метою цього розділу є розробка архітектурних та проектних рішень, які задовольнятимуть сформульовані вимоги та слугуватимуть основою для подальшої реалізації програмного продукту. Проектування буде представлено за допомогою діаграм уніфікованої мови моделювання (UML) [13] та моделей баз даних.

## 2.1 Розробка діаграм поведінки веб-застосунку "Фінансовий Помічник"

### 2.1.1 Розробка діаграм варіантів використання для різних категорій користувачів

Моделювання поведінки системи є важливим етапом проектування, що дозволяє візуалізувати та уточнити взаємодію користувачів із системою та внутрішню логіку виконання її функцій. Це допомагає краще зрозуміти динаміку роботи ПЗ, перевірити повноту та несуперечність функціональних вимог, а також створити чітку основу для розробників та тестувальників. У цьому підрозділі будуть розроблені діаграми варіантів використання (Use Case) для представлення основних сценаріїв взаємодії, а також діаграми активності (Activity) для деталізації реалізації ключових варіантів використання, що є стандартними елементами UML [13].

Діаграма варіантів використання (Use Case Diagram) є одним з основних інструментів для моделювання функціональних вимог з точки зору користувача [13]. Вона показує, які основні дії (варіанти використання) може виконувати користувач (актор) при взаємодії з системою, а також відносини між акторами та варіантами використання.

Відповідно до аналізу, проведеного в розділі 1 (п. 1.3.2), основним та єдиним на даному етапі проектування класом користувачів є "Фінансовий новачок". Саме цей актор буде взаємодіяти з системою для управління своїми особистими фінансами.

На основі функціональних вимог, сформульованих у п. 1.3.3, можна виділити наступні основні варіанти використання для актора "Фінансовий новачок":

* Керувати транзакціями (включаючи додавання, редагування, видалення, імпорт, категоризацію).
* Керувати бюджетами (створення, встановлення лімітів, перегляд виконання).
* Керувати фінансовими цілями (створення, відстеження прогресу).
* Переглядати звіти та аналітику.
* Керувати рахунками.
* Керувати категоріями.
* Зареєструватися в системі.
* Увійти до системи.

Деякі з цих варіантів використання можуть бути пов'язані між собою. Наприклад, операції додавання, редагування та видалення часто є типовими для різних сутностей (транзакції, бюджети, цілі, рахунки, категорії). Це так звані CRUD-операції (Create, Read, Update, Delete). Замість створення окремих Use Cases для кожної операції з кожною сутністю, можна використовувати більш узагальнені варіанти використання, такі як "Керувати транзакціями" або "Керувати бюджетами", як зазначено вище. Деталізація конкретних операцій буде показана на діаграмах уточнення (п. 2.1.2).

Також можна виділити загальні дії, які є частиною інших варіантів використання. Наприклад, "Категоризувати транзакцію" може бути як самостійною дією (при редагуванні), так і обов'язковою частиною "Додати транзакцію вручну" чи "Імпортувати транзакції". У таких випадках доцільно використовувати відношення <<include>>. Відношення <<extend>> може використовуватися для опціональної або виняткової поведінки [13], наприклад, "Обробити помилку імпорту" може розширювати "Імпортувати транзакції".

Діаграма варіантів використання для системи "Фінансовий Помічник" представлена на рисунку 2.1. Вона візуалізує основні взаємодії користувача з системою відповідно до вимог. Для кращої організації діаграми перевага надається горизонтальному розташуванню елементів та використанню достатньо великого шрифту.

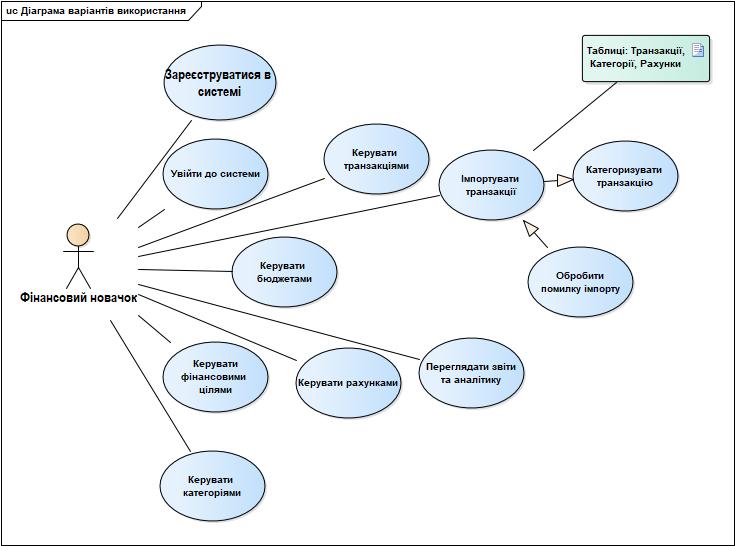


 Рисунок 2.1 - Діаграма варіантів використання системи "Фінансовий Помічник":

Ця діаграма дає загальне уявлення про функціональні можливості системи з точки зору користувача. Однак вона не показує послідовність дій або детальну логіку. Для цього використовуються діаграми уточнення.

### 2.1.2 Уточнення реалізації варіантів використання

Як зазначалося у вимогах, для повного розуміння поведінки системи необхідно деталізувати ключові або складні варіанти використання. Це дозволяє перетворити загальний опис Use Case на послідовність конкретних кроків, які можуть бути реалізовані розробником. Для уточнення можна використовувати різні типи діаграм UML, такі як діаграми активності (Activity Diagram) або діаграми послідовності (Sequence Diagram) [13]. Діаграми активності добре підходять для візуалізації потоку робіт, розгалужень та паралельних дій, тоді як діаграми послідовності краще показують взаємодію між об'єктами (наприклад, користувачем, інтерфейсом, контролером, базою даних) у часі.

У рамках цієї роботи будуть деталізовані два типових варіанти використання за допомогою діаграм активності:

1. "Додати транзакцію вручну": Показує кроки, які виконує користувач та система при введенні нової транзакції через інтерфейс.
2. "Створити бюджет": Ілюструє процес налаштування нового бюджету користувачем.

Діаграми будуть містити основний потік подій, а також можливі альтернативні шляхи (наприклад, обробка помилок валідації). Для чіткого розмежування відповідальності будуть використані доріжки (swimlanes), що представляють Актора ("Фінансовий новачок") та Систему ("Фінансовий Помічник").

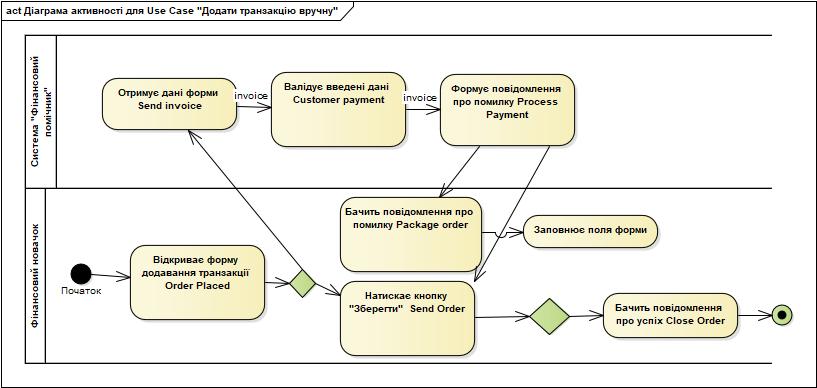


Рисунок 2.22 - Діаграма активності для Use Case "Додати транзакцію вручну"

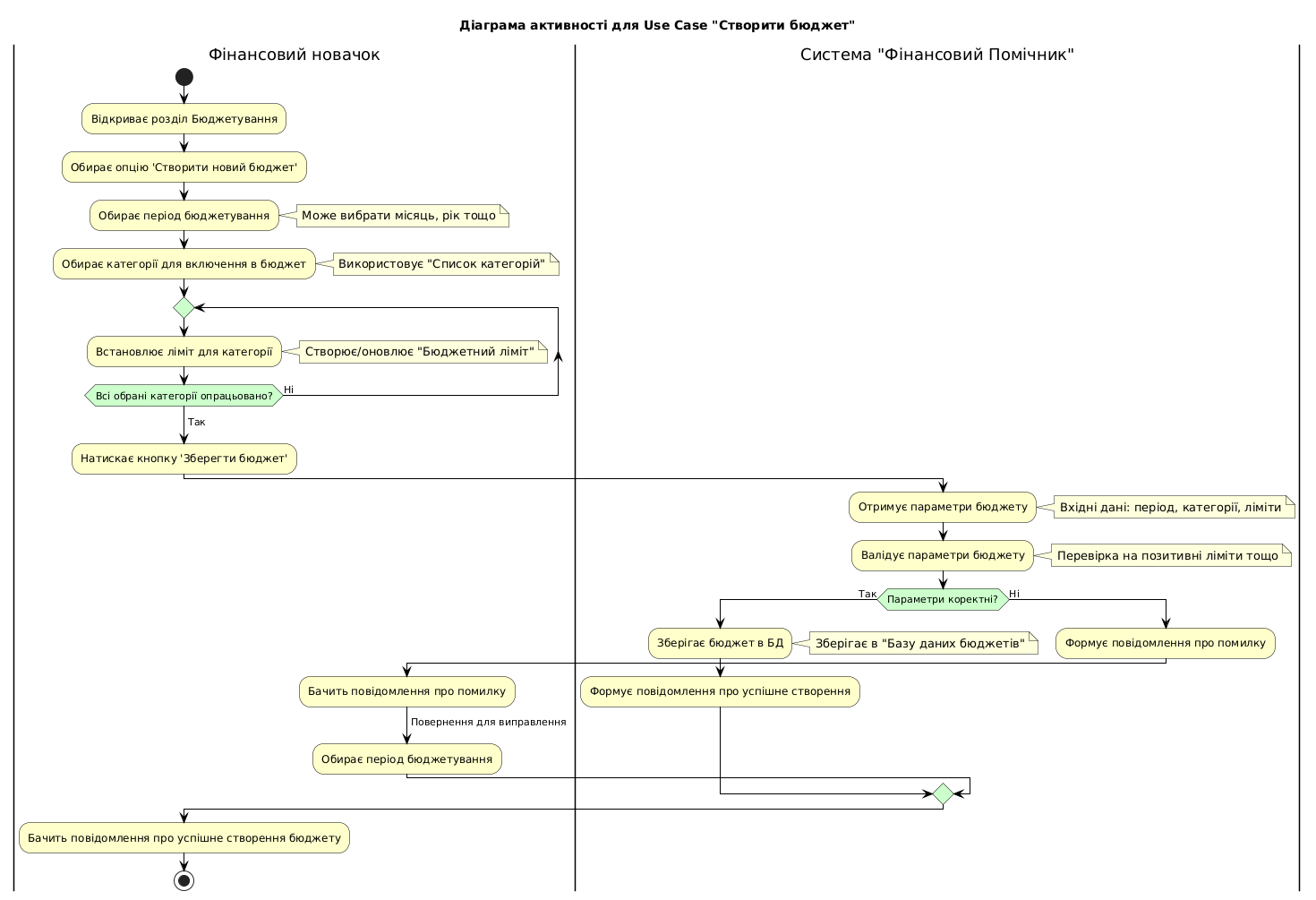


Рисунок 2.3.1 – Діаграма активності для Use Case "Створити бюджет"

Ця діаграма деталізує кроки створення бюджету, включаючи вибір параметрів, встановлення лімітів та збереження даних системою з відповідною валідацією.

## 2.2 Розробка бази даних для зберігання інформації

Після моделювання поведінки системи та уточнення основних варіантів використання (п. 2.1), наступним важливим етапом проектування є визначення структури даних, необхідних для функціонування веб-застосунку "Фінансовий Помічник". Ефективна організація зберігання інформації є критично важливою для забезпечення цілісності даних, продуктивності системи та можливості реалізації всіх функціональних вимог, визначених у розділі 1.3.

Враховуючи обраний технологічний стек (Python/Django), який включає потужний механізм об'єктно-реляційного відображення (ORM), проектування бази даних буде здійснюватися через визначення моделей даних (Entity-класів) на рівні застосунку. Передбачається, що фізична структура бази даних у вибраній СКБД (PostgreSQL) буде згенерована автоматично засобами Django ORM на основі цих моделей. Такий підхід (Code-First) спрощує розробку та підтримку синхронізації між кодом застосунку та схемою БД.

Цей підрозділ включає визначення ключових бізнес-правил, що стосуються даних, та розробку логічної структури даних у вигляді діаграми класів та детального опису їх атрибутів. Принципи реляційного моделювання та забезпечення цілісності даних є основою для проектування надійної бази даних [15].

### 2.2.1 Визначення бізнес-правил предметної області

Перед проектуванням структури даних необхідно сформулювати бізнес-правила та обмеження цілісності, що випливають з предметної області управління особистими фінансами та функціональних вимог. Ці правила визначають допустимі значення даних, взаємозв'язки між ними та обмеження на операції, забезпечуючи коректність та консистентність інформації в системі [15].

Основні бізнес-правила та обмеження для системи "Фінансовий Помічник":

1. Дані користувача:

* Кожен користувач системи повинен мати унікальну адресу електронної пошти (використовується як логін).
* Усі фінансові дані (рахунки, категорії, транзакції, бюджети, цілі) належать конкретному користувачеві і не можуть бути доступні іншим користувачам.

1. Рахунки:

* Назва рахунку має бути унікальною в межах одного користувача.
* Початковий баланс рахунку може бути нульовим або позитивним. Поточний баланс розраховується на основі транзакцій.
* Валюта рахунку є обов'язковим атрибутом (за замовчуванням UAH).
* Неможливо видалити рахунок, якщо з ним пов'язані транзакції.

1. Категорії:

* Назва категорії має бути унікальною в межах одного користувача та одного рівня ієрархії (якщо підтримується ієрархія).
* Кожна категорія має належати до типу "Дохід" або "Витрата".
* Категорія може мати батьківську категорію (для реалізації ієрархії), але не може бути батьківською сама для себе.
* Неможливо видалити категорію, якщо вона використовується у транзакціях або бюджетних лімітах.

1. Транзакції:

* Сума транзакції повинна бути позитивним числом.
* Дата транзакції є обов'язковим атрибутом.
* Кожна транзакція повинна бути обов'язково пов'язана з одним рахунком та однією категорією.
* Тип транзакції (дохід/витрата) визначається типом пов'язаної категорії.

1. Бюджети та Ліміти:

* Бюджет створюється для певного користувача на конкретний період (наприклад, місяць/рік).
* Бюджет складається з одного або кількох лімітів по категоріях витрат.
* Сума ліміту має бути позитивним числом.
* Кожен ліміт у межах одного бюджету пов'язаний з унікальною категорією витрат.

1. Фінансові цілі:

* Назва цілі має бути унікальною в межах одного користувача.
* Цільова сума має бути позитивним числом.
* Бажана дата досягнення є опціональною.
* Ціль може бути пов'язана з певним рахунком для накопичення (опціонально).

Ці правила будуть враховані при проектуванні класів моделі даних та визначенні обмежень на рівні бази даних (через налаштування ORM або безпосередньо в СКБД).

### 2.2.2 Розробка структури бази даних

Структура бази даних для зберігання інформації веб-застосунку "Фінансовий Помічник" буде представлена у вигляді діаграми класів UML [13]. Ця діаграма відображає основні сутності предметної області як класи, їх атрибути (поля) та взаємозв'язки між ними. Оскільки використовується Django ORM, ці класи безпосередньо відповідатимуть моделям у коді застосунку, на основі яких буде згенеровано таблиці в PostgreSQL. Назви класів та атрибутів будуть англійською мовою для зручності розробки, але семантика буде описана українською.

Основні класи (сутності) моделі даних описані в табл.

Таблиця ........

|  |  |
| --- | --- |
| Назва сутності | Семантика |
| UserProfile | Додаткова інформація про користувача (пов'язана зі стандартною моделлю User Django). |
| Account | Рахунки користувача (готівка, картки тощо). |
| Category | Категорії доходів та витрат (з можливістю ієрархії). |
| Transaction | Фінансові операції (доходи та витрати). |
| Budget | Бюджет на певний період. |
| BudgetLimit | Ліміт витрат за конкретною категорією в рамках бюджету. |
| FinancialGoal | Фінансові цілі користувача. |

Діаграма класів, що відображає ці сутності та їх зв'язки, представлена на рисунку 2.4.

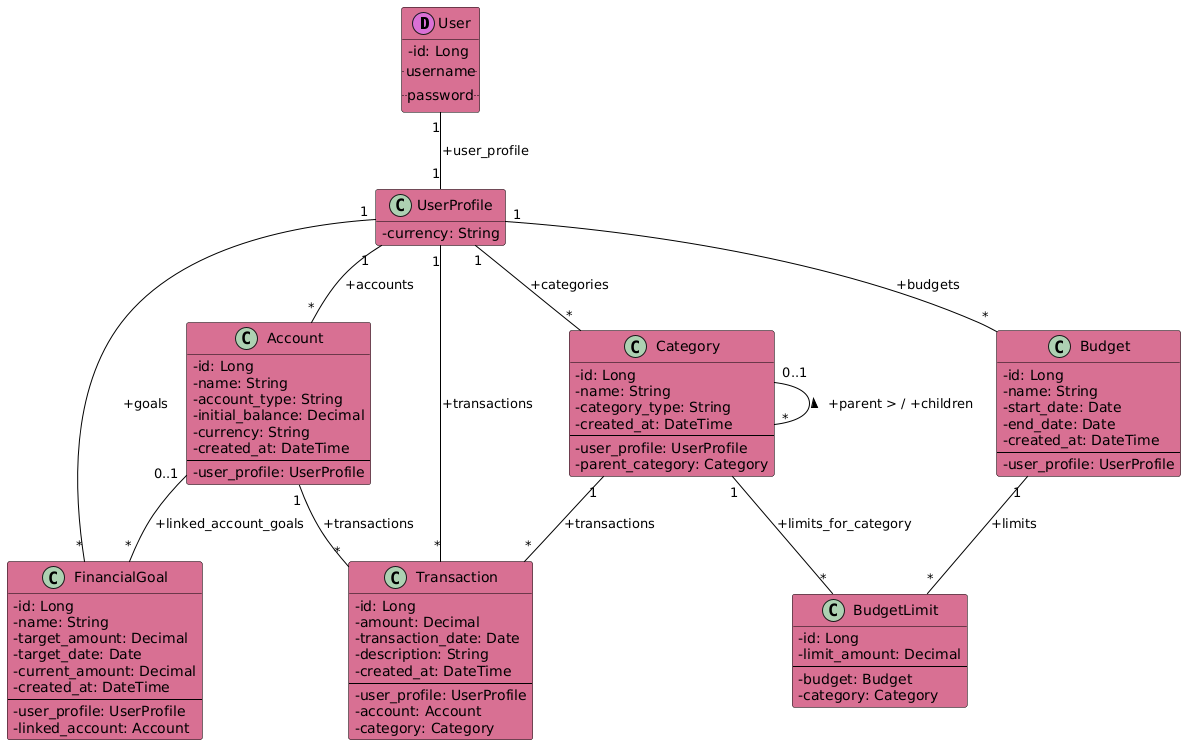


Рисунок 2.4 - Діаграма класів моделі даних

Діаграма відображає логічну структуру даних та основні взаємозв'язки між сутностями системи.

Детальний опис полів (атрибутів) для кожного класу наведено у таблицях нижче (Таблиці 2.1 - 2.7). В описі вказано назву поля, відповідний тип даних (орієнтовно для Python/Django), обмеження та семантичне призначення.

Таблиця 2.1 – Опис полів класу UserProfile

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| user | OneToOneField(User) | PK (неявно), FK, Unique | Зв'язок з вбудованою моделлю користувача Django |
| currency | CharField(max\_length=3) | Not Null | Основна валюта користувача (напр., 'UAH', 'USD') |

Таблиця 2.2 – Опис полів класу Account

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор рахунку |
| user\_profile | ForeignKey(UserProfile) | Not Null | Користувач-власник рахунку |
| name | CharField(max\_length=100) | Not Null, Unique (per user) | Назва рахунку (напр., "Карта Приват", "Готівка") |
| account\_type | CharField(max\_length=20) | Not Null (з вибором) | Тип рахунку (готівка, картка, банк. рахунок тощо) |
| initial\_balance | DecimalField / FloatField | Not Null, Default=0 | Початковий баланс при створенні рахунку |
| currency | CharField(max\_length=3) | Not Null | Валюта рахунку |
| created\_at | DateTimeField(auto\_now\_add=True) | Not Null | Дата та час створення запису |

Таблиця 2.3 – Опис полів класу Category

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор категорії |
| user\_profile | ForeignKey(UserProfile) | Not Null | Користувач-власник категорії |
| name | CharField(max\_length=100) | Not Null, Unique (per user/parent) | Назва категорії (напр., "Їжа", "Транспорт") |
| category\_type | CharField(max\_length=10) | Not Null (з вибором: Дохід/Витрата) | Тип категорії |
| parent\_category | ForeignKey('self', null=True, blank=True) | Nullable | Батьківська категорія (для ієрархії) |
| created\_at | DateTimeField(auto\_now\_add=True) | Not Null | Дата та час створення запису |

Таблиця 2.4 – Опис полів класу Transaction

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор транзакції |
| user\_profile | ForeignKey(UserProfile) | Not Null | Користувач-власник транзакції |
| account | ForeignKey(Account) | Not Null | Рахунок, з яким пов'язана транзакція |
| category | ForeignKey(Category) | Not Null | Категорія, до якої належить транзакція |
| amount | DecimalField / FloatField | Not Null, Positive | Сума транзакції (завжди позитивна) |
| transaction\_date | DateField / DateTimeField | Not Null | Дата (та час) здійснення транзакції |
| description | TextField(null=True, blank=True) | Nullable | Додатковий опис, коментар до транзакції |
| created\_at | DateTimeField(auto\_now\_add=True) | Not Null | Дата та час створення запису |

Таблиця 2.5 – Опис полів класу Budget

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор бюджету |
| user\_profile | ForeignKey(UserProfile) | Not Null | Користувач-власник бюджету |
| name | CharField(max\_length=100) | Not Null | Назва бюджету (напр., "Бюджет на Липень") |
| start\_date | DateField | Not Null | Дата початку бюджетного періоду |
| end\_date | DateField | Not Null | Дата кінця бюджетного періоду |
| created\_at | DateTimeField(auto\_now\_add=True) | Not Null | Дата та час створення запису |

Таблиця 2.6 – Опис полів класу BudgetLimit

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор ліміту |
| budget | ForeignKey(Budget) | Not Null | Бюджет, до якого належить ліміт |
| category | ForeignKey(Category) | Not Null, Unique (per budget) | Категорія витрат, для якої встановлено ліміт |
| limit\_amount | DecimalField / FloatField | Not Null, Positive | Запланована сума ліміту для категорії |

Таблиця 2.7 – Опис полів класу FinancialGoal

|  |  |  |  |
| --- | --- | --- | --- |
| Ім'я поля | Тип даних (Python/Django) | Обмеження | Семантика |
| id | AutoField | PK | Унікальний ідентифікатор цілі |
| user\_profile | ForeignKey(UserProfile) | Not Null | Користувач-власник цілі |
| name | CharField(max\_length=150) | Not Null, Unique (per user) | Назва цілі (напр., "На машину", "Відпустка") |
| target\_amount | DecimalField / FloatField | Not Null, Positive | Сума, яку потрібно накопичити |
| target\_date | DateField(null=True, blank=True) | Nullable | Бажана дата досягнення цілі |
| current\_amount | DecimalField / FloatField | Not Null, Default=0 | Поточна накопичена сума (може оновлюватися) |
| linked\_account | ForeignKey(Account, null=True, blank=True) | Nullable | Рахунок, на якому ведеться накопичення (якщо є) |
| created\_at | DateTimeField(auto\_now\_add=True) | Not Null | Дата та час створення запису |

Ця спроектована структура даних враховує основні вимоги до системи та бізнес-правила, забезпечуючи логічну основу для зберігання та управління фінансовою інформацією користувача. На основі цих класів Django ORM зможе згенерувати відповідні таблиці та зв'язки у базі даних PostgreSQL. Для наочності, результуюча схема таблиць бази даних може бути представлена на окремому рисунку (див. опис рисунку 2.5).

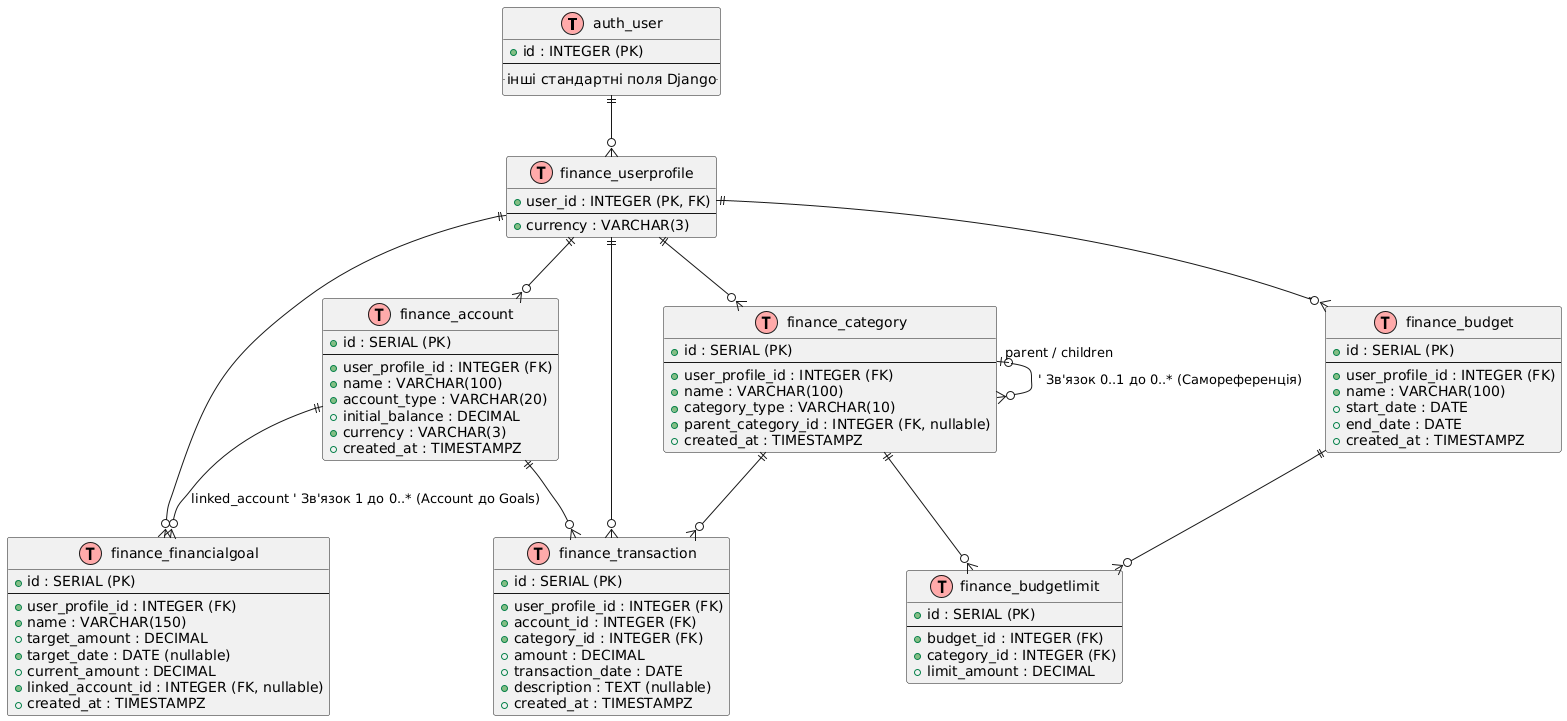


Рисунок 2.5 – Структура таблиць бази даних (Реляційна схема)

Таким чином, проектування бази даних виконано шляхом визначення класів моделі даних, їх атрибутів та зв'язків, а також бізнес-правил, що забезпечують цілісність даних. Це створює надійну основу для зберігання інформації у веб-застосунку "Фінансовий Помічник".

## 2.3 Розробка структурних діаграм програмного забезпечення

Після визначення поведінки системи (п. 2.1) та структури її даних (п. 2.2), важливим етапом є проектування статичної структури самого програмного забезпечення. Це передбачає визначення основних програмних компонентів (класів, модулів, пакетів), їх відповідальностей та взаємозв'язків між ними. Моделювання структури допомагає забезпечити належну організацію коду, сприяє його модульності, підтримуваності та можливості повторного використання [7, 11, 14].

Для візуалізації статичної структури застосунку широко використовується діаграма класів UML [13]. На відміну від діаграми класів моделі даних (рис. 2.4), яка фокусувалася на сутностях, що зберігаються в базі даних, діаграма класів застосунку показує ключові класи, відповідальні за реалізацію бізнес-логіки, обробку запитів користувача, взаємодію з базою даних та представлення даних.

При проектуванні структури веб-застосунку "Фінансовий Помічник" на базі фреймворку Django доцільно дотримуватися архітектурних принципів, що сприяють розділенню відповідальностей. Хоча Django часто асоціюють з патерном MTV (Model-Template-View), для побудови більш гнучких та масштабованих систем, особливо з використанням API (наприклад, за допомогою Django Rest Framework), часто застосовуються підходи, близькі до MVC (Model-View-Controller) або MVVM (Model-View-ViewModel), де логіка чітко розділена між компонентами, відповідальними за дані (Model), їх представлення (View/Template) та обробку логіки (View/Controller/Service).

У рамках даного проекту пропонується виділити наступні логічні шари або групи класів:

Представлення (Views/API): Класи, відповідальні за прийом HTTP-запитів від користувача (через браузер або потенційний мобільний клієнт), обробку цих запитів, виклик відповідної бізнес-логіки та повернення HTTP-відповіді (HTML-сторінки або JSON-дані для API).

Серіалізація (Serializers): (Актуально при використанні API) Класи, що відповідають за перетворення складних типів даних (наприклад, об'єктів моделей Django) у формати, придатні для передачі через API (наприклад, JSON), та навпаки, а також за валідацію даних, що надходять від клієнта.

Бізнес-логіка / Сервіси (Services): Класи або модулі, що інкапсулюють складну бізнес-логіку, яка не належить безпосередньо до моделей даних або обробки запитів. Це може включати розрахунки, перевірки, координацію взаємодії між різними моделями.

Моделі Даних (Models): Класи, що представляють структуру даних, яка зберігається у базі даних (розглянуті детально у п. 2.2.2). Вони також містять логіку, безпосередньо пов'язану з даними (наприклад, методи для отримання пов'язаних об'єктів).

Високорівнева діаграма класів, що ілюструє ці основні компоненти та взаємозв'язки між ними для застосунку "Фінансовий Помічник", представлена на рисунку 2.6.

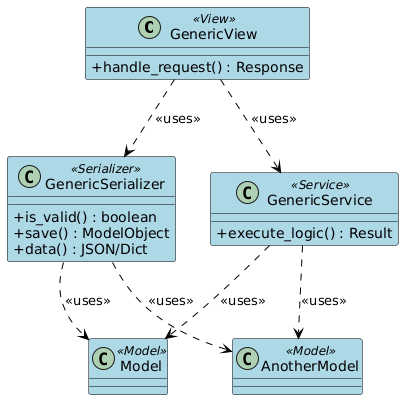


Рисунок 2.6 – Діаграма класів застосунку "Фінансовий Помічник" (Компактна, Високорівнева)

Ця діаграма надає статичний знімок основних програмних компонентів застосунку та їх взаємозалежностей, ілюструючи обраний архітектурний підхід до розділення відповідальностей.