Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій

Лабораторна робота №3

із дисципліни *«Технології розроблення програмного забезпечення»*

Тема: «ДІАГРАМА РОЗГОРТАННЯ. ДІАГРАМА КОМПОНЕНТІВ. ДІАГРАМА ВЗАЄМОДІЙ ТА ПОСЛІДОВНОСТЕЙ»

Виконав:

студент групи ІА-23
Пожар Д. Ю.

Перевірив:

Мягкий М.Ю.

Київ 2024

Тема лабораторних робіт:

Варіант 27

Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)

Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) - на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Розробити діаграму розгортання для проектованої системи.
3. Розробити діаграму компонентів для проектованої системи.
4. Розробити діаграму послідовностей для проектованої системи.
5. Скласти звіт про виконану роботу

Зміст

Хід роботи

Крок 1. Короткі теоретичні відомості

Діаграма розгортання (Deployment Diagram) використовується для візуалізації фізичного розташування системи на пристроях. Вона демонструє, де саме та на якому обладнанні працюють різні частини системи. Ця діаграма допомагає розробникам зрозуміти, як система функціонує на реальних пристроях і як її частини взаємодіють між собою. Основними елементами є вузли (пристрої чи середовища виконання) та артефакти (виконувані файли, конфігураційні файли тощо).

Діаграма компонентів (Component Diagram) відображає структуру системи, розділяючи її на компоненти, які взаємодіють між собою. Вона дозволяє побачити, як система організована на окремі частини, а також візуалізує структуру коду та залежності між компонентами.

Діаграма взаємодії демонструє, як компоненти або об'єкти системи взаємодіють один з одним у рамках певного процесу. Вона ілюструє послідовність обміну повідомленнями між об'єктами в конкретному контексті, відображаючи порядок дій та їх логіку. Ця діаграма зосереджується на переходах між активностями та результатах цих переходів, які можуть змінювати стан системи або повертати певне значення. Основними елементами є об'єкти або учасники, лінії життя (що вказують на тривалість існування об'єкта під час процесу), повідомлення між об'єктами, активності, які вони виконують, та можливі обмеження або умови.

Діаграма послідовностей (Sequence Diagram) ілюструє, як об'єкти у системі взаємодіють між собою з плином часу. Вона показує обмін повідомленнями між учасниками (користувачами, сервером, базою даних) для виконання конкретних задач. Основними елементами цієї діаграми є об'єкти, лінії життя, фокуси управління та повідомлення, що відображають виклик методів чи передачу даних у певній послідовності.

Крок 2. Діаграма розгортання

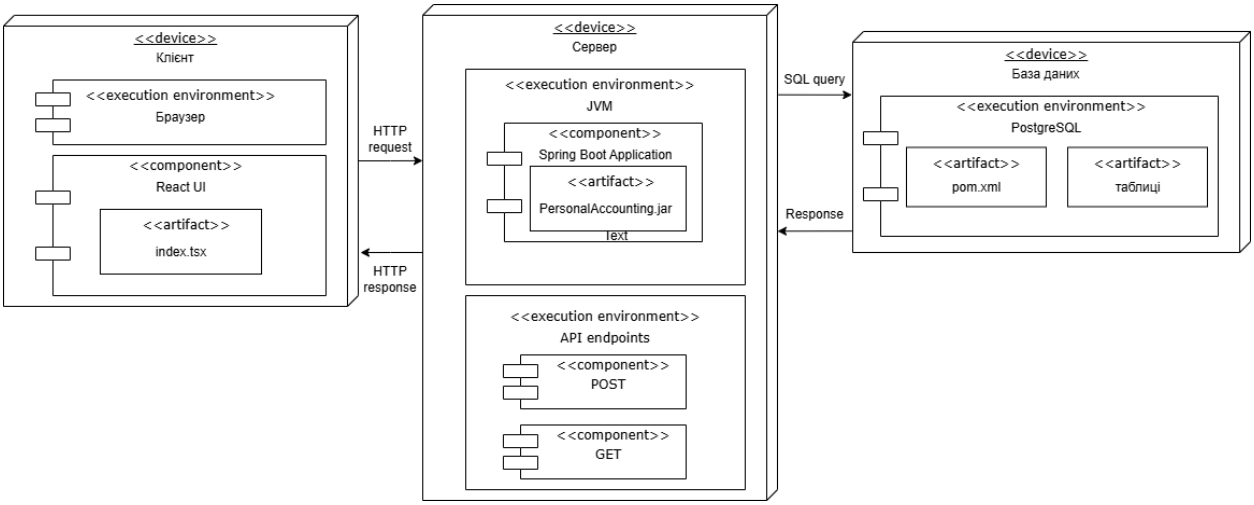


Рис1. Діаграма розгортання

Пристрої (Devices):

Клієнт (<<device>>): це пристрій користувача, який взаємодіє із системою через веб-браузер. На ньому розгорнуті:

- *Середовище виконання* (<<execution environment>>) — **Браузер,** що забезпечує відображення інтерфейсу користувача.

- *Компонент* (<<component>>) — **React UI**, який відповідає за фронтенд частину застосунку. React забезпечує відображення та взаємодію з користувачем.
- *Артефакт* (<<artifact>>) — **index.tsx**, який є основним файлом створення проєкту за допомогою React.

Сервер (<<device>>): Це пристрій, де розгорнута серверна частина системи. На сервері розгорнуті:

- *Середовище виконання JVM* (<<execution environment>>) — **JVM** (Java Virtual Machine), де працює Spring Boot Application.
- *Компонент* (<<component>>) — **Spring Boot Application**, що відповідає за серверну логіку обробки запитів.
- *Артефакт* (<<artifact>>) — **PersonalAccounting.jar**, скомпільований файл, що містить реалізацію серверної частини.

- Додатково розгорнуті **API Endpoints** як середовище виконання, яке містить компоненти:

- **POST** та **GET** компоненти: точки доступу для обробки HTTP-запитів, відповідно для додавання нових даних та отримання існуючих.

База даних (<<device>>): Це пристрій або сервер бази даних, де зберігаються всі дані системи.

- *Середовище виконання* (<<execution environment>>) — **PostgreSQL**, що представляє систему керування базами даних.

- *Артефакт* (<<artifact>>) — **pom.xml**, який є конфігураційним файлом для серверу, містить налаштування роботи бази даних.

- *Артефакт* (<<artifact>>) — **Таблиці**, які є структурою для зберігання даних у базі.

Крок 3. Діаграма компонентів

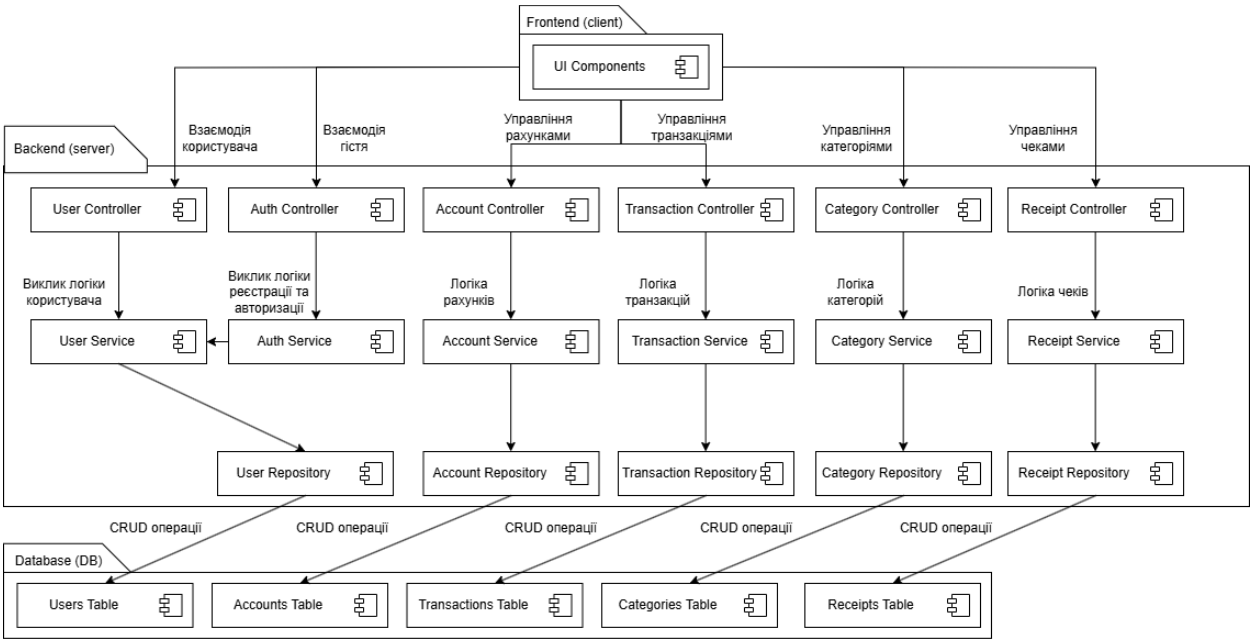


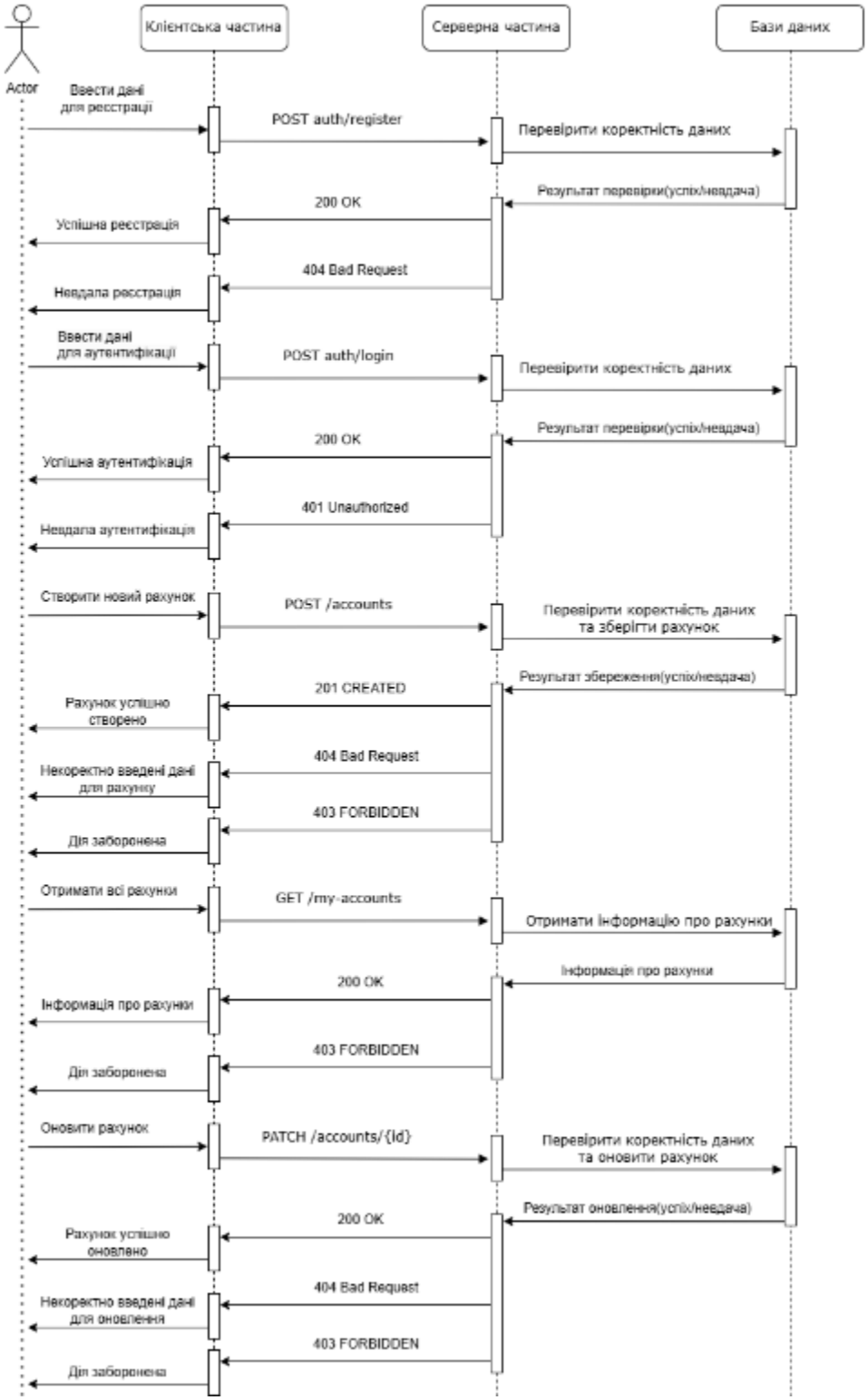
Рис2. Діаграма компонентів

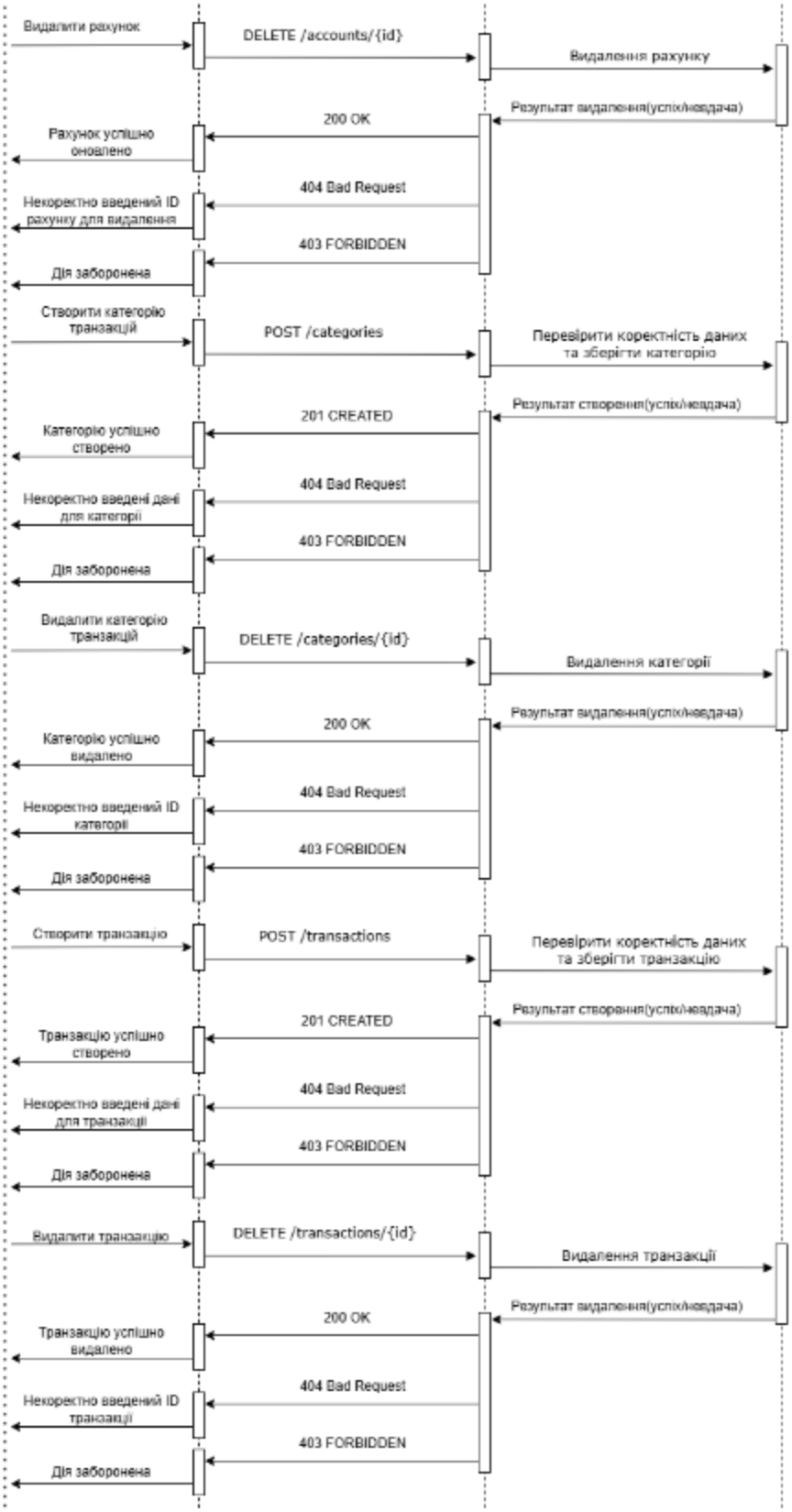
- **UI Component**: Цей компонент відповідає за інтерфейс користувача, який дозволяє користувачам взаємодіяти з системою. Інтерфейс включає функції управління рахунками, транзакціями, категоріями, чеками та профілем користувача.
- **UserController**: Відповідає за обробку запитів, пов'язаних із взаємодією користувача, зокрема отриманням інформації про профіль, його оновленням або видаленням.
- **AuthController**: Реалізує функціонал для реєстрації нових користувачів та аутентифікації вже зареєстрованих користувачів (вхід у систему, вихід).

- **AccountController:** Обробляє запити, пов’язані з рахунками. Наприклад, створення нового рахунку, редагування, видалення або отримання списку всіх рахунків користувача.
- **TransactionController:** Приймає запити для управління транзакціями. Наприклад, додавання нової транзакції, перегляд історії транзакцій, фільтрація за категоріями чи датою.
- **CategoryController**: Відповідає за управління категоріями витрат і доходів. Зокрема, додавання, редагування або видалення категорій.
- **ReceiptController**: Приймає запити, пов’язані з чеками, такі як завантаження чеків, прив’язка їх до транзакцій або перегляд інформації про збережені чеки.
- **UserService**: Реалізує логіку роботи з користувачами: створення, оновлення, видалення профілю.
- **AccountService**: Містить бізнес-логіку для управління рахунками: створення, редагування, підрахунок загального балансу тощо.
- **TransactionService**: Реалізує функціонал управління транзакціями, включаючи їх додавання, редагування, видалення, а також аналітику витрат та доходів.
- **CategoryService**: Відповідає за логіку роботи з категоріями, зокрема забезпечує створення нових категорій, їх редагування та збереження.
- **ReceiptService**: Включає логіку для роботи з чеками, наприклад, розпізнавання тексту чеків, прив’язка чеків до транзакцій, збереження чеків у базі даних.
- **UserRepository**: Забезпечує доступ до таблиці користувачів у базі даних. Виконує CRUD-операції.

- **AccountRepository**: Виконує операції з таблицею рахунків, включаючи створення, оновлення, видалення та отримання даних про рахунки.
- **TransactionRepository**: Взаємодіє з таблицею транзакцій, забезпечуючи доступ до історії витрат і доходів користувача.
- **CategoryRepository**: Здійснює CRUD-операції з таблицею категорій, що зберігає інформацію про типи витрат і доходів.
- **ReceiptRepository**: Відповідає за збереження та отримання даних про чеки у базі даних.
- **Users Table**: Містить інформацію про користувачів, зокрема username, паролі та інші дані профілю.
- **Accounts Table**: Таблиця для збереження даних про рахунки користувача (назва, тип, баланс тощо).
- **Transactions Table**: Містить інформацію про транзакції, включаючи суму, дату, категорію, тип (дохід/витрата) та прив’язку до рахунків.
- **Categories Table**: Зберігає інформацію про категорії витрат і доходів, наприклад "Їжа", "Транспорт", "Зарплата"
- **Receipts Table**: Таблиця для збереження даних про чеки, включаючи зображення, суму та інші деталі.

Крок 4. Діаграма послідовності




Видалити рахунок
DELETE /accounts/{id}
Видалення рахунку
Результат видалення(успіх/невдача)
200 OK
Рахунок успішно оновлено
404 Bad Request
Некоректно введений ID рахунку для видалення
403 FORBIDDEN
Дія заборонена
Створити категорію транзакцій
POST /categories
Перевірити коректність даних та зберігти категорію
Результат створення(успіх/невдача)
201 CREATED
Категорію успішно створено
404 Bad Request
Некоректно введені дані для категорії
403 FORBIDDEN
Дія заборонена
Видалити категорію транзакцій
DELETE /categories/{id}
Видалення категорії
Результат видалення(успіх/невдача)
200 OK
Категорію успішно видалено
404 Bad Request
Некоректно введений ID категорії
403 FORBIDDEN
Дія заборонена
Створити транзакцію
POST /transactions
Перевірити коректність даних та зберігти транзакцію
Результат створення(успіх/невдача)
201 CREATED
Транзакцію успішно створено
404 Bad Request
Некоректно введені дані для транзакції
403 FORBIDDEN
Дія заборонена
Видалити транзакцію
DELETE /transactions/{id}
Видалення транзакції
Результат видалення(успіх/невдача)
200 OK
Транзакцію успішно видалено
404 Bad Request
Некоректно введений ID транзакції
403 FORBIDDEN
Дія заборонена

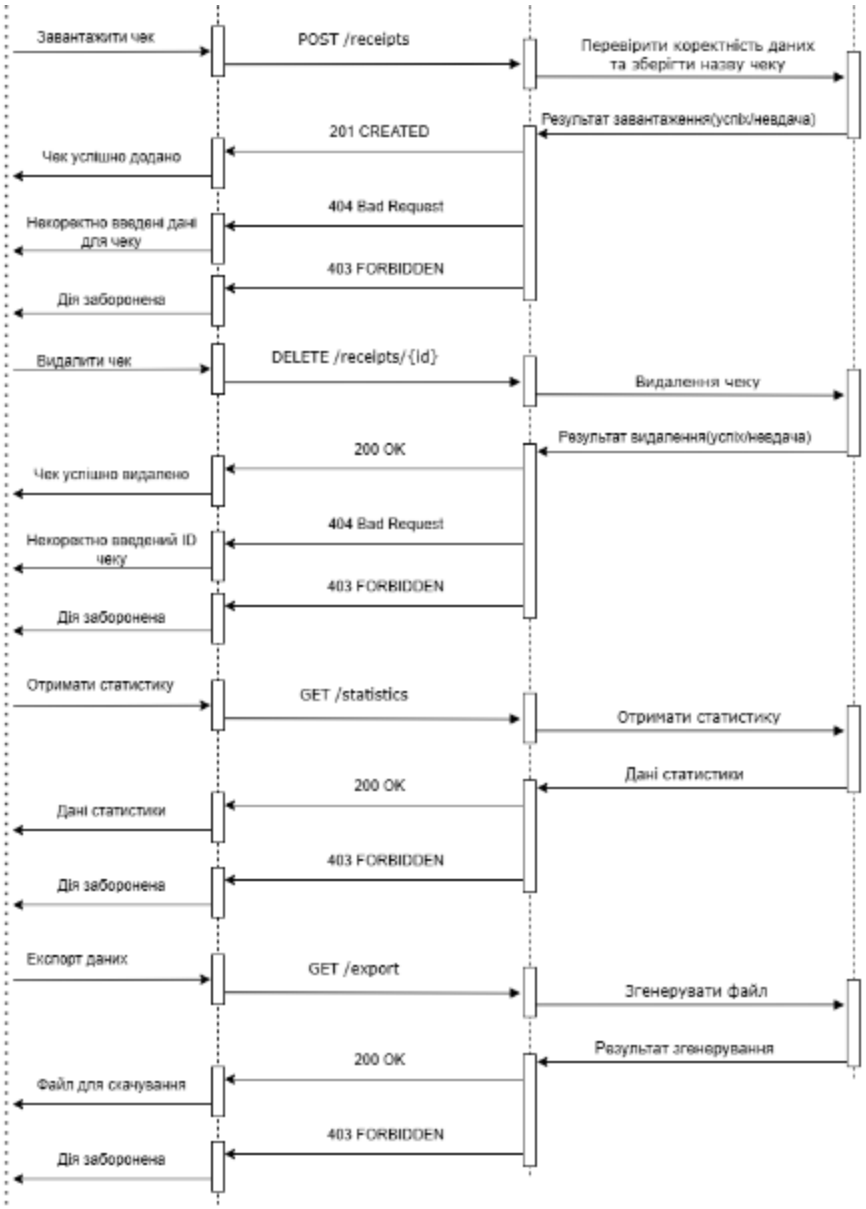


Рис3. Діаграма послідовності

[Посилання на повну діаграму послідовності]

Висновок

У ході лабораторної роботи я ознайомився з теоретичними відомостями про UML-діаграми, що використовуються для моделювання програмних систем.

Було розроблено три основні діаграми для системи "Особиста бухгалтерія. Діаграма розгортання відобразила фізичну структуру системи та взаємодію між сервером, клієнтом і базою даних. Діаграма компонентів показала логічну структуру програмних модулів, таких як управління витратами, статистика, імпорт/експорт в Excel і робота з банківськими рахунками. Діаграма послідовностей моделювала взаємодію об'єктів під час виконання основних функцій системи, таких як введення транзакцій і побудова статистики.

Розроблені діаграми допомогли зрозуміти архітектуру та взаємодію компонентів системи, що сприяло її кращому плануванню і майбутньому впровадженню.