НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____ /Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___» _____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

« Програмне забезпечення для аналізу даних користувачів в онлайн-бібліотеці»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : _____ / Василюк-Зайцева С.В./
підпис ПІБ

Виконав: _____ / Грегуль В.В./
підпис ПІБ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис
"__ __" _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Грегулю Володимиру Валентиновичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи:Програмне забезпечення для аналізу даних користувачів в онлайн-бібліотеці»

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . ___. ___
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____/ Василюк-Зайцева С.В/
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Грегуль В.В. /
підпис ініціали та прізвище

Дата отримання завдання ___ 2024 .

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	- Application Programming Interface — інтерфейс прикладного програмування
CSV	- Comma-Separated Values — формат табличних даних у вигляді тексту
Docker	- Платформа для контейнеризації застосунків
ER-діаграма	- Entity-Relationship Diagram — діаграма сутність-зв'язок
Excel	- Програмне забезпечення Microsoft для обробки таблиць
FastAPI	- Python-фреймворк (бібліотека високого рівня) для створення API
Google Sheets	- Онлайн-таблиці Google
HTML	- HyperText Markup Language — мова розмітки гіпертексту
HTTP	- HyperText Transfer Protocol протокол передачі гіпертексту
HTTPS	- Безпечний варіант протоколу HTTP
JavaScript	- Мова програмування для веб-інтерфейсів
JSON	- JavaScript Object Notation — текстовий формат для структурованих даних
JWT	- JSON Web Token — формат токена для аутентифікації
Matplotlib	- Python-бібліотека для побудови графіків
npm	- Node Package Manager — менеджер пакетів для JavaScript
OpenAPI	- Специфікація для опису REST API
Pandas	- Python-бібліотека для обробки та аналізу даних
PDF	- Portable Document Format — формат електронних документів
Postman	- Інструмент для тестування API-запитів
PostgreSQL	- Об'єктно-реляційна система керування базами даних
Power BI	- Платформа для створення звітів та візуалізацій від Microsoft

Pydantic	- Python-бібліотека для перевірки даних на основі типів
Python	- Мова програмування загального призначення
React.js	- JavaScript-бібліотека для побудови інтерфейсів
Render	- Платформа для хостингу застосунків
RESTful API	- Архітектурний стиль побудови API на основі HTTP
Seaborn	- Бібліотека Python для статистичної візуалізації
SQLAlchemy	- ORM-бібліотека для Python
Swagger UI	- Інтерфейс для інтерактивної документації OpenAPI
UML	- Unified Modeling Language — уніфікована мова моделювання
Vercel	- Платформа хостингу для інтерфейсної частини застосунків
Vite	- Інструмент для швидкої розробки інтерфейсної частини системи
XLSX	- Формат таблиць Microsoft Excel

ВСТУП

В епоху глобальної цифровізації електронні бібліотеки стали невід'ємною складовою інформаційного простору, забезпечуючи доступ до знань та культурної спадщини. З огляду на постійне збільшення аудиторії цих ресурсів, диверсифікацію користувацьких запитів та підвищені стандарти обслуговування, виникає нагальна потреба в удосконаленні механізмів керування цифровими бібліотечними фондами та веб сайтами. Зокрема, нагального значення набуває розробка спеціалізованих програмних рішень для моніторингу та аналізу читацької активності. Такі інструменти дають змогу не тільки зібрати дані про користувацьку поведінку, але й використовувати їх для стратегічного планування розвитку контенту, вдосконалення інтерфейсу та створення персоналізованих сервісів на основі проаналізованих вихідних даних.

Вибрана тематика дипломного дослідження обумовлена гострою необхідністю автоматизувати аналітичні процеси в онлайн-бібліотеках. В умовах швидкого зростання інформаційних потоків, традиційні методи опрацювання даних втрачають ефективність, що породжує запит на інноваційні технологічні рішення для збирання, опрацювання, узагальнення та візуалізації статистичних показників та даних. Впровадження таких систем не лише підвищує достовірність аналітичних висновків, але й забезпечує оперативне коригування контенту відповідно до змін читацьких уподобань, дозволяє розробникам оптимізувати структури електронних ресурсів та підвищити якость обслуговування користувачів онлайн бібліотек. Крім того, подібні аналітичні інструменти сприяють обґрунтуванню довгострокових стратегій щодо модернізації контентної політики та технічної інфраструктури цифрових бібліотек.

Головне завдання дипломної роботи полягає у створенні веб-орієнтованого аналітичного застосунку, який надає бібліотечному аналітику інструментарій для безпосередньої роботи з користувацькими даними: їх імпорту, автоматизованого аналізу активності, генерування аналітичних звітів, представлення результатів у зручному форматі та експорт звітів. Функціональність розробленої системи охоплює: відстеження статистичних показників, виявлення пріоритетних літературних жанрів і авторів, розрахунок тривалості користувацьких сесій, профілювання аудиторії, а також графічне відображення виявлених трендів і кореляцій.

Для втілення проєкту застосовано комплекс сучасних технологічних рішень: серверну частину реалізовано мовою Python із використанням високорівневої бібліотеки FastAPI, система управління даними базується на PostgreSQL, а обробка інформації здійснюється за допомогою бібліотек Pandas, SQLAlchemy та Pydantic. Інтерфейсну частину ПЗ розроблено на основі React.js із застосуванням збірника Vite. Програмний комплекс забезпечує підтримку імпорту даних у форматах JSON та CSV, здійснює їх автоматичну перевірку та опрацювання, створює візуалізації за допомогою Matplotlib [13] і Seaborn [21], а також підтримує експорт результатів у різноманітних форматах (PDF, CSV, XLSX, JSON). Безпека доступу гарантується механізмом JWT-автентифікації, а API-інтерфейс документовано відповідно до стандарту OpenAPI через Swagger UI [15].

Система призначена для роботи одного користувача — аналітика, та забезпечує самостійне виконання повного циклу аналітичних операцій без залучення технічних спеціалістів. Архітектура рішення характеризується гнучкістю та масштабованістю, а розгортання відбувається у контейнеризованому середовищі Docker. Тестування функціональності

проводилось із використанням Postman, інтерактивної документації FastAPI та методології сценарного тестування в браузері. Інтерфейсну складову розміщено на платформі Vercel [24], а серверну частину — на Render [7].

Структура пояснювальної записки включає чотири розділи, які послідовно висвітлюють системний аналіз предметної області, методологію проєктування та реалізації програмного забезпечення, а також практичні аспекти впровадження й експлуатації системи. Загальний обсяг роботи складає XXX сторінок, містить 32 ілюстрацій, 11 таблиць, 11 фрагментів програмного коду та 31 бібліографічних джерел.

Отже, створена система повністю відповідає сучасним викликам у сфері цифрового бібліотечного менеджменту та є прикладом інтегрованого, практично-спрямованого рішення, що гармонійно поєднує інноваційні технології, високу продуктивність та інтуїтивно зрозумілий інтерфейс.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У двадцять першому столітті, цифрові бібліотеки відіграють дедалі важливішу роль як джерело знань, інформації та культурної спадщини. Вони надають користувачам доступ до великого обсягу літератури, наукових публікацій, періодики та іншого контенту в електронному вигляді. При цьому, як і будь-який цифровий сервіс, онлайн-бібліотека накопичує велику кількість даних про дії своїх користувачів, що можуть бути використані для покращення якості надання послуг, підвищення залученості читачів та оптимізації внутрішніх процесів.

Для глибшого розуміння специфіки розробленого програмного забезпечення необхідно визначити ключові елементи предметної області:

Користувач онлайн-бібліотеки — фізична особа, яка взаємодіє з цифровим ресурсом через особистий кабінет або гостьовий доступ. Користувачі можуть здійснювати перегляд книг, зберігати улюблені видання, залишати відгуки, здійснювати пошук за назвою чи жанром та ділитися знайденою інформацією з іншими користувачами.

Дані про активність користувачів — структурована інформація про взаємодію користувача з бібліотекою: жанрові вподобання, частота входу в систему, історія пошукових запитів, оцінки книг. Ці дані зазвичай зберігаються у вигляді записів або файлів у форматі JSON у базі даних.

Аналітик — особа, яка здійснює аналіз зібраних даних з метою формування звітів та виявлення поведінкових закономірностей. В умовах онлайн-бібліотеки аналітик може бути співробітником IT-відділу, керівником проєкту або спеціалістом із цифрової трансформації.

Файл з даними — електронний документ, що містить записані події або атрибути користувачів. Наприклад, файл формату JSON може включати список сесій користувача із зазначенням дати, дій, тривалості, використаного пристрою тощо.

Звіт — структурований документ, що є результатом аналізу даних. Звіт може містити статистичні показники, графіки, діаграми, зведення активності за період, а також частоту повторного використання сервісу.

1.2 Основні процеси предметної області

Для аналізу даних користувачів в онлайн-бібліотеці було визначено низку бізнес-процесів, що утворюють логіку функціонування системи. Їх реалізація дозволяє забезпечити повний цикл аналітичної обробки — від збору первинних даних до формування готових статистичних звітів (див. табл. 1.1).

1.2.1 Завантаження даних

Процес завантаження даних є першим етапом роботи з інформацією про активність користувачів онлайн-бібліотеки. Він охоплює такі послідовні етапи:

- Авторизація аналітика: користувач-аналітик авторизується в системі за допомогою своєї електронної пошти та паролю. Після успішної аутентифікації, система генерує JWT-токен, що гарантує захищений доступ до функцій системи та даних, які належать зареєстрованому аналітику.
- Завантаження файлу даних: аналітик завантажує отриманий ним файл у форматі JSON, що містить дані про активність користувачів. Для завантаження використовується спеціалізований модуль FastAPI, який отримує файл через HTTP-запит і зберігає його на сервері [9].
- Перевірка цілісності та коректності даних: після завантаження система автоматично перевіряє сумісність файлу з алгоритмом обробки даних,

реалізованим у системі. Також система перевіряє наявність обов’язкових полів, таких як ідентифікатор користувача (реалізована логіка цього відрізку коду розташована у Додатку Б), дата й час дій, інформація про переглянуті книги та виконує аналіз на наявність дублювання записів, відсутність критичних помилок та правильність форматування даних.

- Збереження інформації у базу даних: у разі успішного проходження перевірки інформація про завантажений файл зберігається у базу даних PostgreSQL. Кожен файл прив’язується до відповідного аналітика через унікальний ідентифікатор, що забезпечує простий подальший доступ та управління завантаженими даними.
- Повідомлення аналітика про статус завантаження: після завершення процесу система сповіщає користувача про результати завантаження. У випадку помилок надається чітке повідомлення із зазначенням причин, що дозволяє скорегувати файл і повторити спробу.

1.2.2 Автоматизована обробка даних

Після успішного завантаження файлу система автоматично агрегує отримані записи в єдину структуровану базу даних. На основі цих даних створюються таблиці, які відображають інформацію про поведінку користувачів, такі як перегляди книг, пошукові запити та рейтинги:

Автоматично здійснюється обчислення ключових метрик, таких як:

- Загальна кількість користувачів.
- Найпопулярніші жанри, книги та автори.
- Рейтингова оцінка контенту за відгуками користувачів.

На етапі фільтрації даних проводиться виявлення та видалення дублюючих записів, які можуть штучно збільшити вагу окремих дій користувачів і негативно вплинути на об'єктивність статистичного аналізу (див. Додаток Б).

Після завершення очищення даних відбувається сегментація користувачів. Система після нормалізації завантаженихї даних, автоматично групує користувачів за різноманітними критеріями, такими як вік, професію, рівень освіти, тип пристрою, що використовується для доступу до бібліотеки, а також жанрові вподобання. Така сегментація дозволяє більш глибоко дослідити поведінкові характеристики різних категорій читачів, виявити типові сценарії їхньої взаємодії з платформою, а також забезпечити персоналізацію подальших рекомендацій до потреб цільової аудиторії.

Останнім етапом аналізу є підготовка структурованих таблиць на основі готових звітових даних. Система автоматично формує агреговані таблиці, що містять узагальнену інформацію за ключовими параметрами: активність користувачів, динаміка відвідувань, найпопулярніші жанри, тощо. Такі таблиці створюються у форматі, зручному для швидкого перегляду, подальшого використання у автоматизованих звітах, а також візуалізації даних у вигляді графіків та діаграм.

1.2.3 Візуалізація даних

Візуалізація даних дозволяє швидко і наочно представити великі обсяги інформації у вигляді простих діаграм та графіків, які були раніше підготовлені програмним забезпеченням для подальшої візуалізації. Завдяки візуальному представленню даних аналітик може виявити тренди, аномалії, кореляції та закономірності, які складно виявити при роботі лише з табличними, текстовими або файловими форматами.

1.2.4 Експорт результатів

Після завершення процесу аналітичної обробки даних та формування звітів і візуалізацій, система надає користувачеві можливість експортувати створені звіти у зручних форматах для подальшого використання.

Експорт реалізовано у вигляді генерації файлів форматів PDF, CSV, JSON та XLSX, що дає змогу адаптувати матеріали як для зовнішньої презентації, так і для глибшого подальшого аналізу в середовищах типу Excel, Power BI, Python або Google Sheets.

Таблиця 1.1

Основні бізнес-процеси предметної області

№	**Назва процесу**	**Опис**
1	2	3
1	Завантаження даних	Імпорт вхідного файлу користувача, перевірка структури та збереження
2	Обробка даних	Очистка, агрегація, нормалізація даних для аналітичної обробки
3	Формування звітів	Побудова звітів за параметрами, вибір періоду, форматування виводу
4	Візуалізація статистики	Побудова графіків, діаграм, аналітичних таблиць
5	Експорт результатів	Збереження звіту в локальному або зовнішньому форматі

Джерело: сформовано автором

1.3 Моделювання предметної області

UML (Unified Modeling Language) — уніфікована мова моделювання, що використовується розробниками програмного забезпечення для візуалізації процесів та роботи систем [3]. Вона дозволяє спростити процеси представлення, документування та передачі інформації про структуру, поведінку та взаємодію програмних систем. Завдяки використанню UML, розробники можуть

створювати наочні візуалізації, що допомагають краще розуміти, аналізувати та комунікувати архітектуру системи, її компоненти, зв’язки та процеси [3].

Для моделювання різних типів систем, таких як програмні додатки або бази даних, UML надає набір діаграм, які охоплюють різні аспекти архітектури та поведінки системи. Ці діаграми використовуються для візуалізації структурних компонентів, взаємодії між елементами системи або для опису процесів, що відбуваються в межах системи.

1.3.1 Діаграма прецедентів

Діаграма варіантів використання — (діаграма прецедентів, сценарій використання, use case) — дозволяє уявити типи ролей та їх взаємодію із системою. Проте не показує порядок виконання кроків. Зображує функціональні вимоги (те, що система може зробити) з точки зору користувача [3].

У нашій системі передбачено лише одного користувача — аналітика, який взаємодіє з програмним забезпеченням і виконує всі основні функції, такі як завантаження даних, аналіз активності користувачів, формування звітів та візуалізація результатів (див. рис. 1.1). Усі ці функції визначають варіанти використання системи, що надають аналітику усі необхідні інструменти для проведення аналізу.

Зокрема, варіанти використання, зображені на діаграмі, включають:

- Завантаження файлів — аналітик може завантажувати файли з даними, що підлягають аналізу.
- Керування файлами — це можливість редагувати чи видаляти файли з даними.
- Обробка даних та створення звітів — аналітик може проводити аналіз даних і створювати відповідні звіти.

- Створення та експорт звітів — після аналізу, система дозволяє генерувати звіти та експортувати їх у різних форматах.
- Перегляд статистики — аналітик має змогу переглядати статистику активності користувачів і популярність контенту як і на головній панелі програми, так і на сторінки аналітики.

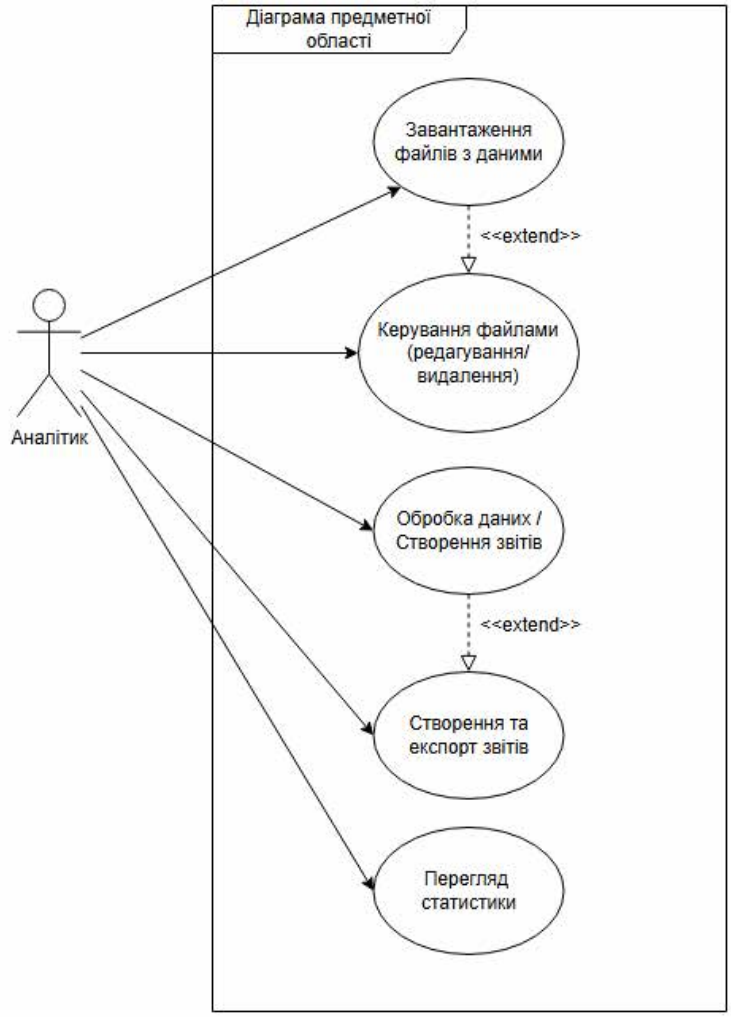


Рисунок 1.1 – Діаграма прецедентів предметної області

1.3.2 Абстракція предметної області

Абстракція предметної області подає узагальнену модель взаємодії основних об'єктів у системі аналізу даних користувачів онлайн-бібліотеки. Вона дозволяє сформувати концептуальне уявлення про ролі, дані та процеси, які

підтримуються програмним забезпеченням, і слугує базою для побудови детальніших логічних і фізичних моделей.

У контексті розробленої системи виокремлюються такі ключові компоненти:

Аналітик (Користувач) — основний суб'єкт взаємодії із системою. Його функції охоплюють авторизацію, завантаження файлів з даними, ініціювання аналітичної обробки, перегляд результатів, а також експорт згенерованих звітів. Кожен аналітик має унікальний ID, ім'я та електронну пошту та зашифрований пароль.

Файл даних — вхідний об'єкт, який містить структуровану інформацію про активність користувачів бібліотеки. Кожен файл має унікальний ідентифікатор, дату завантаження, тип (формат) та прив'язується до конкретного аналітика. Він слугує джерелом для подальшого аналізу.

Звіт — агрегований результат аналізу даних, сформований на основі одного або кількох файлів. Звіт містить ключові метрики, графіки, таблиці, виявлені закономірності та може бути збережений для повторного використання.

Експорт звіту — завершальний етап взаємодії із системою, який дозволяє користувачу зберегти сформований звіт у зручному форматі (PDF, CSV, JSON, XLSX) для подальшої презентації або аналітичної обробки в інших системах.

Бібліотека (дані про активність користувачів) — не є безпосереднім учасником взаємодії, але виступає джерелом інформації, що імпортується до системи у вигляді файлів. З погляду абстракції, бібліотека виконує роль видавника даних.

На основі цієї абстракції можна окреслити загальний потік процесу:

A. Бібліотека надає дані про активність користувачів у форматі файлів.
B. Аналітик завантажує ці файли до системи.

C. Система виконує автоматизовану обробку: очищення, агрегацію, побудову звітів.

D. Результати аналізу виводяться у вигляді звітів і візуалізацій.

E. Експорт дозволяє аналітику зберегти результати у потрібному форматі.

Нижче зображена абстракція (див. рис. 1.2) є підґрунтям для формалізації бізнес-логіки системи, побудови класів, таблиць бази даних і структури взаємодії через API.

Рисунок 1.2 – Абстракція предметної області

1.3 Аналіз вимог до програмної системи

Процес визначення вимог до програмної системи необхідний для чіткого визначення функціональних можливостей, якими повинна володіти система для ефективного виконання поставлених завдань. Я виділив наступні вимоги до системи своєї розроблюваної системи: функціональні, нефункціональні, вимоги до інтерфейсу користувача та вимоги до програмного інтерфейсу.

1.3.1 Функціональні вимоги

Функціональні вимоги визначають, які функції повинна виконувати програмна система для задоволення потреб користувачів (див. у табл. 1.2).

Таблиця 1.2

Функціональні вимоги до програмного забезпечення для аналізу даних користувачів в онлайн бібліотеці

№	**Вимога**	**Опис**
1	2	3
1	Збір даних про активність користувачів	Програмне забезпечення повинно мати можливість отримувати дані про активність користувачів онлайн-бібліотеки, зокрема перегляди, пошукові запити, оцінки, додавання до обраного, завантаження книг та інші дії.
2	Завантаження та обробка даних	Система повинна підтримувати завантаження файлів у форматах JSON, здійснювати їх перевірку на коректність та відповідність структурі, а також автоматично очищати та сегментувати дані для подальшого аналізу.
3	Аналіз даних та формування звітів	Після обробки даних система повинна автоматично генерувати аналітичні звіти. Це включає побудову статистики користувацької активності, популярності

		жанрів і авторів, а також сегментації користувачів за різними критеріями.
4	Візуалізація даних	Програмне забезпечення повинно дозволяти створювати графіки та діаграми для візуалізації статистичних даних. Це реалізовано у вигляді лінійних діаграм, кругових графіків тощо.
5	Експорт результатів	Користувач має можливість експортувати звіти у різні формати, такі як PDF, CSV, JSON або XLSX, для подальшого аналізу або використання в інших системах.

Джерело: створено автором

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги - це характеристики програмної системи, які не стосуються безпосередньо її функціональності, але важливі для забезпечення її ефективності, зручності використання та надійності. Нефункціональні вимоги до розроблюваної системи зазначені у таблиці 1.3.

Таблиця 1.3

Нефункціональні вимоги до програмного забезпечення для аналізу даних користувачів в онлайн бібліотеці

№	**Вимога**	**Опис**
1	2	3
1	Продуктивність	Система повинна обробляти великі обсяги даних без значних затримок. Час відповіді на запит для генерації звіту не повинен перевищувати 5 секунд, а для побудови графіків — 3 секунди.
2	Маштабованість	Програмне забезпечення повинно бути здатне масштабуватися відповідно до зростання обсягу даних, забезпечуючи підтримку великих баз даних з мільйонами записів без втрати продуктивності.
3	Безпека	Система повинна забезпечувати захищений доступ до персональних даних користувачів через використання сучасних методів

		аутентифікації та авторизації (наприклад, JWT [11] для підтвердження особи користувача), а також захищати передавання даних через HTTPS.
4	Надійність	Програмне забезпечення повинно бути стійким до помилок, а також мати систему відновлення даних після збою. Це гарантує, що навіть у разі технічних проблем користувачі не втратять важливу інформацію.

Кінець таблиці 1.3

1	2	3
5	Кросплатформеність	Система повинна бути кросплатформеною, тобто доступною для використання на різних операційних системах (Windows, macOS, Linux).

Джерело: створено автором на основі

1.3.3 Вимоги до інтерфейсу користувача

Дизайн інтерфейсу користувача відіграє важливу роль у створенні ефективних і привабливих веб-сайтів. UI-дизайн відповідає за те, як користувачі взаємодіють з інтерфейсом, які емоції вони відчувають і наскільки легко користуватися сайтом або додатком [2]. Для досягнення високого рівня зручності та ефективності використання, інтерфейс повинен відповідати низці вимог, які забезпечують інтуїтивно зрозуміле управління та адаптивність до різних умов. У випадку з програмним забезпеченням для аналізу даних користувачів онлайн-бібліотеки, ці вимоги охоплюють кілька основних концептів, таких як адаптивний дизайн (responsive design) та доступний дизайн (accessible design) [2].

А. Інтуїтивність та простота використання

Інтерфейс повинен бути простим і зрозумілим навіть для користувачів, які не мають спеціальних технічних навичок. Це означає, що основні функції системи повинні бути легко доступі користувачу не залежно від його пристрою чи версії браузера. Наприклад, система повинна забезпечувати швидкий доступ

до основних інструментів аналізу даних, таких як завантаження файлів, налаштування параметрів звітів та візуалізацій, а також експорт результатів.

Щоб забезпечити інтуїтивність інтерфейсу, слід реалізувати наступні пункти:

- Логічна ієрархія елементів інтерфейсу.
- Використання стандартних і зрозумілих позначень (наприклад, значки для завантаження, фільтрації або налаштування параметрів).
- Підказки та допоміжні тексти для полегшення використання функцій.

B. Адаптивний дизайн (Responsive Design)

Адаптивний дизайн є основною вимогою для сучасних веб-додатків, оскільки він дозволяє забезпечити коректне відображення інтерфейсу на різних пристроях з різними розмірами екрану, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони. Веб-система повинна автоматично підлаштовуватися під розмір екрану користувача, зберігаючи при цьому функціональність і зручність користування.

Це включає:

- Адаптивне розташування елементів: елементи інтерфейсу повинні адаптуватися до різних розмірів екранів, змінюючи свою позицію або масштабування. Наприклад, таблиці або графіки повинні зменшуватися або збільшуватися в залежності від екрану.
- Мобільна версія інтерфейсу: на мобільних пристроях повинна бути зручна навігація з великими кнопками, оптимізованими для дотику, а також доступ до всіх основних функцій, таких як перегляд та завантаження звітів.

C. Доступність інтерфейсу (Accessible Design)

Доступність інтерфейсу — дозволяє зробити програмне забезпечення доступним для користувачів з обмеженими можливостями. Це включає забезпечення функціоналу для людей з порушеннями зору, слуху або моторики.

Ключові аспекти доступності:

- Використання кольорових контрастів: інтерфейс повинен забезпечувати достатній контраст між фоном та текстом, щоб користувачі з порушеннями зору могли легко читати інформацію.
- Підтримка екранних читалок: програмне забезпечення повинно бути сумісним з екранними читалками, що дозволяє користувачам з вадами зору отримувати інформацію через голосові інтерфейси.
- Клавіатурні скорочення та доступність без миші: інтерфейс повинен дозволяти здійснювати основні операції через клавіатуру, без необхідності використовувати мишу, що є важливим для користувачів з обмеженою моторною активністю.

Доступність описана у стандартах WCAG (Web Content Accessibility Guidelines), що надають чіткі рекомендації щодо поліпшення доступності веб-контенту для людей з інвалідністю [25].

D. Взаємодія з користувачем (UX)

Система повинна має бути не лише функціональною, а й зручною для користувача. Тобто, UX повинен в себе включати:

- Швидку реакцію інтерфейсу: система повинна працювати без затримок, забезпечуючи швидке відображення результатів пошуку, завантаження даних та побудови графіків.
- Зворотний зв'язок: система повинна надавати користувачеві зворотний зв'язок у разі успішного виконання дії (наприклад, сповіщення про успішне завантаження файлу або створення звіту).

1.3.4 Вимоги до API

Інтерфейс програмування додатків (API) є “мозком” серверної частини програмного забезпечення, воно дозволяє забезпечити інтеграцію з іншими системами та сторонніми додатками. У випадку розробки програмного забезпечення для аналізу даних користувачів онлайн-бібліотеки API має виконувати низку вимог, які забезпечать ефективну взаємодію між компонентами системи, а також з іншими потенційними користувачами та розробниками.

API повинно бути добре документованим, що забезпечить простоту інтеграції та використання його іншими розробниками чи сторонніми системами. Документація повинна бути зрозумілою і містити чіткі вказівки щодо доступних кінцевих точок програми, методів запитів, форматів даних, що використовуються, а також опис можливих помилок, які можуть виникнути під час виконання запитів [20]. Це дозволить стороннім розробникам без труднощів інтегрувати API в свої системи або автоматизувати процеси взаємодії з даними.

Інтерфейс програмування повинен бути здатен працювати з різними форматами даних, такими як JSON, XML і CSV, що дозволить гнучко обробляти запити і відповідати на них у форматі, зручному для користувачів. Зокрема, для систем, які працюють із великими обсягами даних, підтримка таких форматів є важливою, оскільки дозволяє інтегрувати API з іншими базами даних, додатками та сервісами. Підтримка таких форматів як JSON також забезпечить зручну взаємодію з клієнтськими додатками, що можуть обробляти дані для візуалізації або аналізу.

Одним із найважливіших аспектів є забезпечення високої безпеки API. Для цього необхідно впровадити механізми аутентифікації та авторизації, такі як використання API ключів або токенів доступу, які дозволяють контролювати, хто може взаємодіяти з API і отримувати доступ до чутливих даних. Всі запити до

API повинні здійснюватися через захищені канали зв’язку (HTTPS) [20], щоб гарантувати безпеку передачі даних.

API також повинно підтримувати ефективну обробку запитів з мінімальними затримками. Це означає, що час відповіді на стандартні запити не повинен перевищувати 500ms, а на більш складні запити — 2-3 секунди. Це дозволяє зберігати швидкість та ефективність роботи системи навіть при високих навантаженнях, що є важливим аспектом для інтеграції API в реальних умовах [1] [20].

В решті решт, API повинно мати можливість підтримувати різні методи запитів, зокрема GET, POST, PUT, DELETE [9], що дозволяє отримувати, додавати, оновлювати та видаляти дані з бази. Це дає змогу інтегрувати API в різні системи, де можуть бути необхідні різні типи операцій.

Використання чітких стандартів для організації кінцевих точок та обробки запитів дозволить забезпечити високий рівень сумісності з іншими системами та бібліотеками. Зокрема, потрібно дотримуватись RESTful підходу, що дозволяє організувати API на основі стандартних методів HTTP [9] [15] [20].

1.4 Огляд інформаційних джерел та існуючих рішень

Існує ряд програмних рішень, які допомагають автоматизувати та оптимізувати аналіз даних користувачів онлайн-бібліотек. Це включає як загальні аналітичні платформи, так і спеціалізовані рішення, орієнтовані на бібліотечну діяльність. У даному розділі ми розглянемо деякі з них, їхні переваги та недоліки, а також можливості інтеграції в існуючі бібліотечні системи.

1.4.1 Koha

Koha є однією з найпопулярніших систем для автоматизації бібліотечних процесів з відкритим вихідним кодом. Вона забезпечує управління каталогами,

обслуговуванням користувачів, а також аналітику щодо використання бібліотечних ресурсів. Koha є комплексною системою для управління бібліотечними фондами, але також пропонує можливості для збору та аналізу даних про активність користувачів. Вона дозволяє бібліотекам здійснювати аналіз популярності книг, оцінок користувачів та відгуків [14].

Переваги:

- Відкрите джерело і безкоштовне використання.
- Висока гнучкість налаштувань і можливість інтеграції з іншими інструментами.
- Підтримка детальних звітів щодо активності користувачів і бібліотечних ресурсів.
- Можливість налаштування складних аналітичних звітів.

Недоліки:

- Вимагає технічних знань для налаштування та обслуговування.
- Обмежена підтримка для інтеграції з сучасними зовнішніми аналітичними інструментами.
- Деякі функції можуть бути не такими зручними в порівнянні з більш сучасними платформами для аналітики.

1.4.2 Alma

Alma — це хмарна платформа, яка забезпечує повний цикл управління бібліотечними ресурсами, включаючи збір і аналіз даних про користувачів. Система пропонує потужні можливості для аналізу даних, таких як використання книг, активність користувачів і популярність ресурсів. Alma дозволяє створювати персоналізовані звіти та аналізи за різними параметрами, що робить її популярною серед великих бібліотек і університетів [12].

Переваги:

- Підтримка масштабованих і складних бібліотечних систем.
- Інтеграція з іншими бібліотечними та зовнішніми системами для обміну даними.
- Можливості для детального аналізу користувацької активності та попиту на контент.
- Підтримка персоналізованих звітів, що дає змогу враховувати специфічні потреби користувачів.

Недоліки:

- Висока вартість, що може бути недоступно для малих бібліотек.
- Потрібен досвід для налаштування та підтримки.
- Обмеження щодо кастомізації деяких елементів інтерфейсу і функціональності.

1.4.3 OpenBiblio

OpenBiblio — являє собою систему для управління бібліотечними ресурсами з відкритим вихідним кодом. Вона орієнтована на малих та середніх бібліотек, пропонуючи базові функціональності для автоматизації каталогізації, управління користувачами та аналізу активності. Система дозволяє бібліотекам ефективно працювати з бібліографічними даними і надає основні інструменти для звітності [16].

Переваги:

- Простота використання та налаштування.
- Повністю безкоштовне програмне забезпечення з відкритим вихідним кодом.
- Зручний інтерфейс для малих бібліотек і користувачів без технічних знань.

Недоліки:

- Обмежені функції для глибокого аналізу даних користувачів.

- Не підтримує складні звіти та інтеграцію з сучасними аналітичними інструментами.
- Відсутність деяких просунутих функцій, доступних в більших платформах.

1.4.4 Tableau

Це потужний інструмент для візуалізації даних, який дозволяє створювати інтерактивні графіки та діаграми для аналізу інформації. Хоча Tableau не є специфічно орієнтованим на бібліотеки, він може бути інтегрований для аналізу даних про користувачів онлайн-бібліотек. Завдяки широким можливостям візуалізації та інтеграції з різними джерелами даних, Tableau може використовуватися для створення звітів щодо користувацької активності, аналізу популярності контенту і прогнозування попиту [6].

Переваги:

- Потужні можливості для візуалізації даних та інтеграції з різними джерелами.
- Легкість у створенні інтерактивних звітів та діаграм.
- Підтримка гнучкої налаштування і візуалізації складних даних.

Недоліки:

- Висока вартість ліцензії, що може бути проблемою для малих бібліотек.
- Потрібні спеціалізовані знання для налаштування та оптимізації.

Аналіз існуючих рішень для автоматизації збору та аналізу даних користувачів онлайн-бібліотек показав, що на ринку є широкий вибір програмних платформ, які варіюються від простих інструментів до складних комплексних систем. Кожне з існуючих рішень має свої переваги та недоліки, що робить їх більш чи менш підходящими в залежності від розміру бібліотеки, бюджету та специфічних вимог до функціональності.

Koha та OpenBiblio є відмінними варіантами для малих бібліотек, оскільки вони забезпечують базову автоматизацію бібліотечних процесів і є безкоштовними. Однак їхні можливості щодо аналізу даних користувачів обмежені, що може стати перешкодою для великих бібліотечних систем, де потрібен більш детальний аналіз та інтеграція з іншими інструментами.

Alma і Tableau, з іншого боку, надають потужні можливості для глибокого аналізу даних і інтеграції з іншими системами, але їх висока вартість і складність налаштування роблять їх менш доступними для малих бібліотек. Tableau особливо добре підходить для візуалізації даних, але потребує спеціалізованих знань для налаштування, що може бути не завжди зручно.

Розроблене програмне забезпечення для аналізу даних користувачів онлайн-бібліотеки пропонує оптимальне поєднання простоти використання та потужних аналітичних можливостей. Воно дозволяє автоматизувати всі етапи обробки даних — від завантаження файлів до створення детальних звітів і візуалізацій. Особливістю нашої системи є підтримка гнучкої налаштування звітів та інтеграція з іншими бібліотечними інструментами, що дозволяє адаптувати систему під різні потреби та вимоги користувачів. Завдяки простому інтерфейсу користувач може легко виконувати складні аналітичні задачі без необхідності мати глибокі технічні знання.

Наша система вирізняється також доступністю та адаптивністю для різних типів бібліотек — від малих до великих організацій, що робить її універсальним інструментом для аналізу даних. Вона дає змогу швидко отримати необхідну інформацію про поведінку користувачів, популярність книг, а також відслідковувати зміни в активності читачів, що в свою чергу дозволяє оптимізувати роботу бібліотеки та підвищити якість обслуговування користувачів.

Таким чином, розроблене програмне забезпечення має великий потенціал для подальшого використання в онлайн-бібліотеках, оскільки поєднує високу функціональність, простоту використання та можливості для налаштування під конкретні потреби організації. Це рішення є ефективним інструментом для збору та аналізу даних користувачів, що дозволяє бібліотекам підвищувати рівень обслуговування та орієнтуватися на потреби своєї аудиторії, що особливо важливо в умовах цифрової трансформації.

1.5 Постанова завдання

При розробці програмного забезпечення для аналізу даних користувачів онлайн-бібліотеки основна увага повинна бути зосереджена на вирішенні ключових завдань, пов'язаних з автоматизацією збору, обробки та аналізу даних, а також зручності використання системи для кінцевих користувачів.

Основною проблемою, яку вирішує запропоноване рішення, є відсутність централізованого інструменту для ефективного аналізу великої кількості даних про активність користувачів онлайн-бібліотеки. Сучасні системи часто не надають достатньої гнучкості для проведення детального аналізу поведінки користувачів або візуалізації статистичних даних. Це може призводити до неефективного використання бібліотечних ресурсів та неправильних управлінських рішень.

Задля подолання цієї проблеми пропонується розробка програмного забезпечення, яке автоматизує обробку даних про користувачів онлайн-бібліотеки та дозволяє генерувати аналітичні звіти. Це включатиме розробку інтерфейсу для завантаження даних, їх автоматичної обробки, візуалізації результатів у вигляді графіків і діаграм, а також експорт звітів для подальшого використання. Платформа повинна забезпечити швидку і точну обробку великих

обсягів даних, а також створення гнучких звітів, що відповідають специфічним вимогам бібліотеки.

Програмне забезпечення має на меті підвищити ефективність управління бібліотечними ресурсами, покращити взаємодію з користувачами та підвищити якість надання послуг. Система повинна включати в себе модулі для аналізу популярності книг, вподобань користувачів, а також зібрання інформації про найпоширеніші жанри та авторів. Завдяки автоматизації процесу збору даних та їх аналізу, бібліотеки отримають потужний інструмент для більш точного прогнозування потреб своїх користувачів та оптимізації контенту.

Важливим аспектом є також створення інтуїтивно зрозумілого інтерфейсу для аналітиків, який дозволить зручно завантажувати файли, налаштовувати параметри звітів та отримувати необхідні результати. Звітність та візуалізація даних допоможуть швидко оцінити ефективність роботи бібліотеки та інтереси читачів. Платформа повинна також підтримувати експорт результатів у різних форматах, таких як PDF, CSV, JSON та XLSX, що дозволить зберігати дані для подальшого аналізу або передачі керівництву.

Програмне забезпечення повинно включати функції автоматичного тестування та оптимізації для забезпечення безперебійної роботи системи, навіть при обробці великих обсягів даних. Тестування повинно охоплювати всі аспекти функціональності системи, включаючи перевірку даних, швидкість генерації звітів та правильність їх виведення.

Розробка даної системи дозволить оптимізувати процеси управління бібліотечними ресурсами, забезпечить аналітиків необхідними інструментами для аналізу даних та підвищить загальну ефективність роботи онлайн-бібліотек. У підсумку, це дозволить приймати більш обґрунтовані управлінські рішення,

покращити взаємодію з користувачами та адаптувати бібліотечні ресурси під актуальні потреби аудиторії.

РОЗДІЛ 2 ПРОЄКТУВАННЯ РОЗРОБЛЮВАННОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура розроблюваного програмного забезпечення

Моделювання предметної області дозволяє правильно визначити структуру даних, основні процеси та взаємодії між компонентами системи. У даному розділі буде здійснено опис предметної області для системи аналізу даних користувачів онлайн-бібліотеки, включаючи ідентифікацію основних сутностей, їхніх зв’язків, а також основних процесів взаємодії користувача з системою. Таке моделювання дозволяє створити ясне уявлення про те, як організована система, а також забезпечує основу для подальшої розробки програмного забезпечення.

2.1.1 Визначення основних сутностей

У попередньому розділі (пункт. 1.1) було надано загальне уявлення про ключові елементи предметної області, такі як користувач системи (аналітик), файл із даними, звіт та результати експорту. У цьому підпункті наведено технічну деталізацію цих сутностей, яка є основою для подальшого проєктування архітектури програмного забезпечення.

У системі для аналізу даних користувачів онлайн-бібліотеки передбачено такі основні програмні сутності:

- Користувач (аналітик) — особа, яка взаємодіє з системою з метою аналізу даних. До атрибутів належать: унікальний ідентифікатор (analyst_id), ім’я, електронна пошта та зашифрований пароль. Аналітик може завантажувати файли, формувати звіти й експортувати їх.
- Файл даних — джерело інформації для аналізу, що завантажується аналітиком. Зберігає назву файлу, шлях до нього, дату завантаження та прив’язку до конкретного користувача.

- Звіт — результат обробки завантажених даних. Містить агреговану статистику, побудовану на основі дій користувачів. Основні атрибути: ідентифікатор, назва, дата створення та JSON-структура з метриками.
- Експорт звіту — кінцевий файл, що формується на основі звіту у форматі PDF, CSV, JSON або XLSX. Зберігає інформацію про формат, дату створення та шлях до збереженого файлу.

Ці сутності становлять логічну основу всієї системи, визначають її функціональну структуру та використовуються як при побудові бази даних, так і в логікі роботи API.

2.1.2 Визначення зв’язків між сутностями

Встановлення взаємозв'язків між сутностями дає змогу зрозуміти характер взаємодії між компонентами системи, що забезпечує цілісність інформаційних даних. Окреслені взаємозв'язки між ключовими сутностями системи:

- Користувач і Файл даних

Користувач (аналітик) може завантажити кілька файлів. Це зв’язок "один до багатьох", де кожен аналітик може мати кілька файлів даних, що йому належать. Зв'язок реалізується через зовнішній ключ «analyst_id» в сутності «Data File».

- Користувач і Звіт

Аналітик створює кілька звітів, що відображають результати аналізу даних. Це також зв’язок "один до багатьох", оскільки один аналітик може створити багато звітів. Зв'язок між сутностями визначається через атрибут «analyst_id» в сутності «Analysis Report».

- Звіт і Експорт звіту

Кожен звіт може бути експортований у кілька форматів. Це зв’язок "один до багатьох", де один звіт може мати кілька експортованих файлів. Зв'язок

реалізовано через «report_id» в сутності «Report Export», що пов’язує експортовані файли з конкретними звітами.

Ці зв’язки формують реляційну структуру даних, забезпечуючи цілісність і правильність взаємодії між основними компонентами системи.

2.1.3 Діаграма сутність-зв’язок

На рисунку 2.1 наведена ER діаграма (діаграма сутність-зв’язок) до нашого реалізованого ПЗ. Ця діаграма візуалізує зв’язки між основними сутностями, такими як Користувач, Файл даних, Звіт та Експорт звіту та основою для проєктування бази даних та забезпечує зрозуміле представлення того, як інформація буде зберігатися в системі.

Рисунок 2.1 – ER діаграма предметної області

2.2 Збереження даних

Для організованого та безпечного збереження даних та файлів користувачів, важливо забезпечити правильну організацію бази даних. Для цього буде використовуватися реляційна база даних PostgreSQL, яка є надійним інструментом для зберігання великих обсягів структурованих даних.

PostgreSQL це система керування базами даних корпоративного класу з відкритим кодом. Він підтримує як SQL, так і JSON для реляційних і нереляційних запитів для розширюваності та відповідності SQL. PostgreSQL підтримує розширені типи даних і функції оптимізації продуктивності, які доступні лише в дорогих комерційних базах даних, наприклад Oracle і SQL Server [4]. Саме тому вона гарно підходить для розроблюваного ПЗ, адже основні очікувані дані для збереження та роботи будуть у форматі JSON. У Додатку А зображено SQL код для створення описаної бази даних.

Система буде зберігати інформацію про користувачів (аналітиків), файли з даними, а також результати їх аналізу у вигляді звітів.

2.2.1 Опис діаграми класів

Діаграма класів відображає структуру реалізованих об’єктів системи, їхні атрибути та методи і взаємозв’язки між ними. У контексті системи для аналізу даних користувачів онлайн-бібліотеки, діаграма класів визначає, як зберігаються та взаємодіють компоненти системи. Опис класів, що входять до діаграми класів позначено у таблиці 2.1.

Таблиця 2.1

Класи системи аналізу даних користувачів онлайн бібліотеки

№	**Клас**	**Атрибути**	**Методи**
1	2	3	4

1	Analyst (Аналітик)	analyst_id: ідентифікатор аналітика. analyst_name: ім’я аналітика. email: електронна пошта. hashed_password: зашифрований пароль для безпеки.	upload_data(): метод для завантаження файлів. create_report(): метод для створення звітів. delete_data() для видалення завантажених файлів

Кінець таблиці 2.1

1	2	3	4
2	DataFile (Файл даних)	file_id: унікальний ідентифікатор файлу. filename: ім’я файлу. file_path: шлях до файлу на сервері. upload_date: дата завантаження файлу.	validate_file(): перевірка коректності файлу перед завантаженням. get_file_data(): метод для отримання даних з файлу для подальшої обробки.
3	AnalysisReport (Звіт)	report_id: унікальний ідентифікатор звіту. analyst_id: зв’язок із аналітиком, який створив звіт. report_name: назва звіту. report_data: дані звіту (у форматі JSON).	generate_report(): метод для створення звіту на основі завантажених даних. export_report(): метод для експорту звіту в потрібному форматі (PDF, CSV, JSON).

		created_at: дата створення звіту.	delete_report() для видалення звіту
4	ReportExport (Експорт звіту)	export_id: унікальний ідентифікатор report_id: ідентифікатор звіту. export_format: формат експорту file_path: шлях до файлу. created_at: дата створення експорту.	save_export(): збереження файлу експорту на сервері. get_export_info(): отримання інформації про експорт. delete_export() – для видалення експорту

Джерело: створено автором

2.2.2 Діаграма класів

Діаграма класів для нашої системи включає всі описані вище класи та їхні взаємозв’язки. На діаграмі зображено взаємодію класів та даних у проєктованій системі(див рис. 2.2).

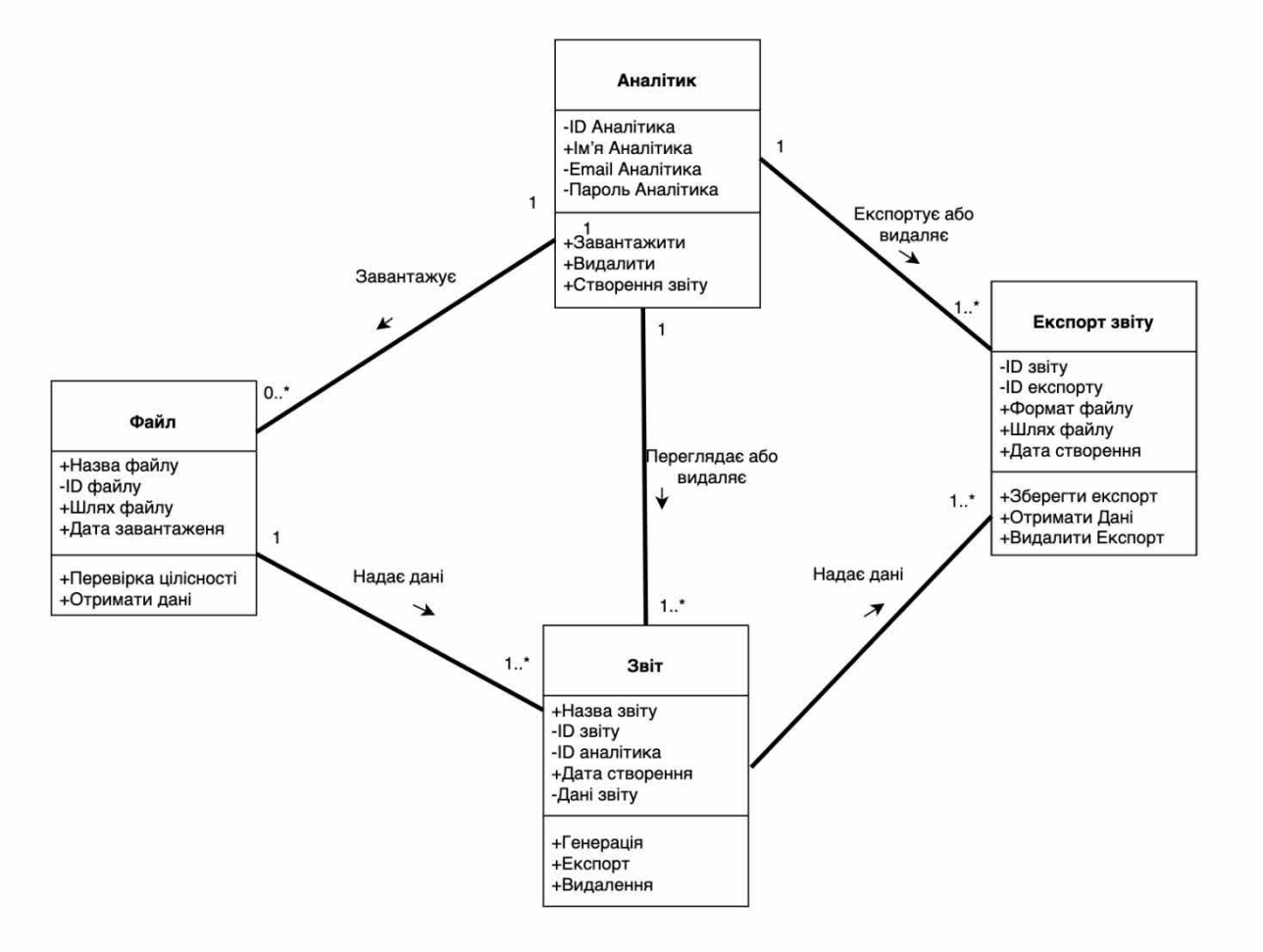


Рисунок 2.2 - Діаграма класів системи для аналізу даних користувачів онлайн-бібліотеки

2.3 Користувацький інтерфейс

Графічний інтерфейс системи надає користувачеві можливість взаємодіяти з програмним забезпеченням за допомогою зручних графічних елементів. Інтерфейс повинен бути інтуїтивно зрозумілим, зручним і ефективним для виконання всіх основних функцій, таких як завантаження даних, обробка інформації, формування звітів та їх експорт.

У цьому підрозділі ми розглянемо основні аспекти побудови інтерфейсу користувача, включаючи схему навігації та діаграму активності, які відображають всі процеси взаємодії користувача з системою.

2.3.1 Профіль потенційного користувача

Основна функція інтерфейсу користувача полягає в наданні аналітику зручного інструменту для завантаження, обробки, візуалізації та експорту статистики за заздалегідь заданими параметрами. Оскільки наша система орієнтована на аналітиків, важливо розуміти, хто є основним користувачем, які його потреби, а також як забезпечити максимально зручний і інтуїтивно зрозумілий інтерфейс для виконання професійних завдань.

Основні характеристики користувача:

- Вік: 25-45 років
- Навички: аналітик повинен мати базові знання в обробці даних, зокрема в роботі з таблицями та статистичними звітами. Він може мати досвід роботи в аналізі даних або економіці, але не обов'язково володіти глибокими технічними знаннями, такими як програмування або адміністрування серверів.
- Освіта: ступінь в галузі економіки, інформаційних технологій, статистики або суміжних спеціальностей.
- Технічні навички: вміння працювати з аналітичними інструментами, такими як Excel або спеціалізовані програми для обробки даних. Основні знання в роботі з таблицями в CSV та JSON форматах.

Потреби користувача:

- Легкість в завантаженні файлів та обробці великих обсягів даних.
- Можливість швидко отримувати звіти та візуалізації на основі наданих даних.

- Простий інтерфейс для роботи без необхідності додаткових технічних знань.

На основі цього профілю розроблено інтерфейс, який забезпечує максимально простий і доступний шлях до виконання основних функцій (див. табл. 2.2).

Таблиця 2.2

Функціональні можливості інтерфейсу для аналітика

№	Функція	Опис
1	2	3
1	Авторизація	Авторизація або реєстрація аналітика у систему.
2	Завантаження файлів	Завантаження файлів у форматі JSON та автоматична перевірка завантаженого файлу на коректність структури даних.
3	Обробка даних	Збирання даних з файлу для подальшого аналізу та створення графіків.
4	Візуалізація даних	Створення графіків і діаграм, проведення аналізу з оброблених даних.
5	Експорт звітів	Експорт результатів аналізу даних у форматах: PDF, CSV, JSON, XLSX.

Джерело: сформовано автором

Ця таблиця ілюструє функціональні можливості інтерфейсу, які були реалізовані з урахуванням потреб аналітика, що допомагає йому швидко та ефективно працювати з даними та отримувати аналітичні звіти.

Інтерфейс користувача включає кілька основних екранів: Авторизація, Основна сторінка, Завантаження файлів, Аналіз даних, Експорт звітів. Користувач може швидко переміщатися між ними завдяки зручній боковій навігаційній панелі (див. рис. 2.3).

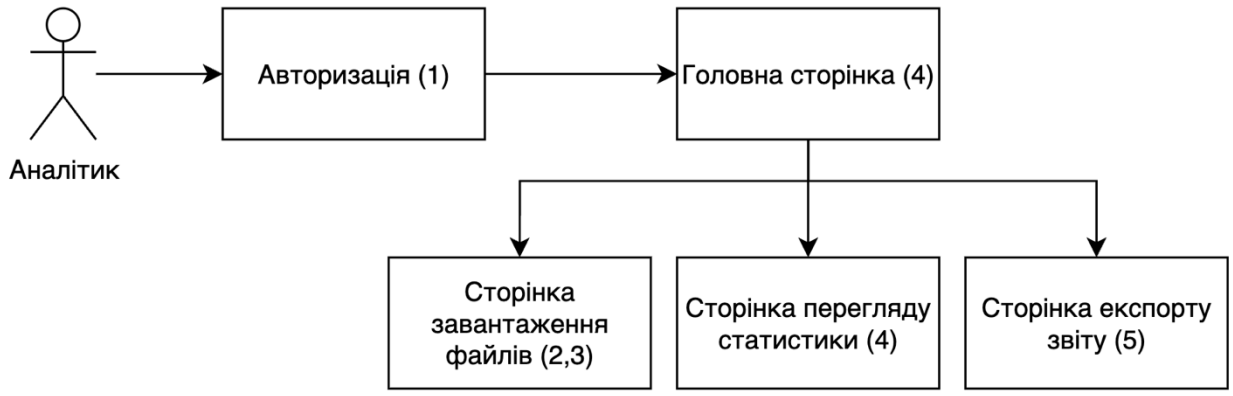


Рисунок 2.3 – Блок-схема навігаційної схеми користувача

Ця блок-схема демонструє, як користувач може переміщатися між екранами для виконання основних завдань: авторизації, завантаження файлів, аналізу даних, перегляду результатів та експорту звітів.

Саме тому, інтуїтивно зрозумілий інтерфейс для аналітика дозволяє йому ефективно виконувати свої завдання з аналізу даних без потреби в додаткових технічних знаннях. В наступному розділі (розділ 3) ми детальніше розглянемо нюанси створення якісного інтерфейсу для користувача.

2.3.2 Діаграми активності

В цьому розділі представлені діаграми активності для основних процесів користувача при використанні розроблюваної системи, а саме:

- Реєстрація та авторизація — користувач спочатку реєструється (якщо це новий користувач) або авторизується в системі через введення свого логіну та пароля (див. рис. 2.4 та рис. 2.5).
- Завантаження файлу з даними — після входу в систему аналітик може завантажити файли з даними для подальшого аналізу. Цей процес включає перевірку правильності формату файлів і їх перевірку (див. рис. 2.6).

- Аналіз даних — після завантаження файлу система обробляє дані, виконуючи необхідні операції для збору статистики і створення звітів (див. рис. 2.6).
- Експорт звіту — після завершення аналізу користувач може експортувати результати у різних форматах (PDF, CSV, JSON) для подальшого використання або презентації (див. рис. 2.7).

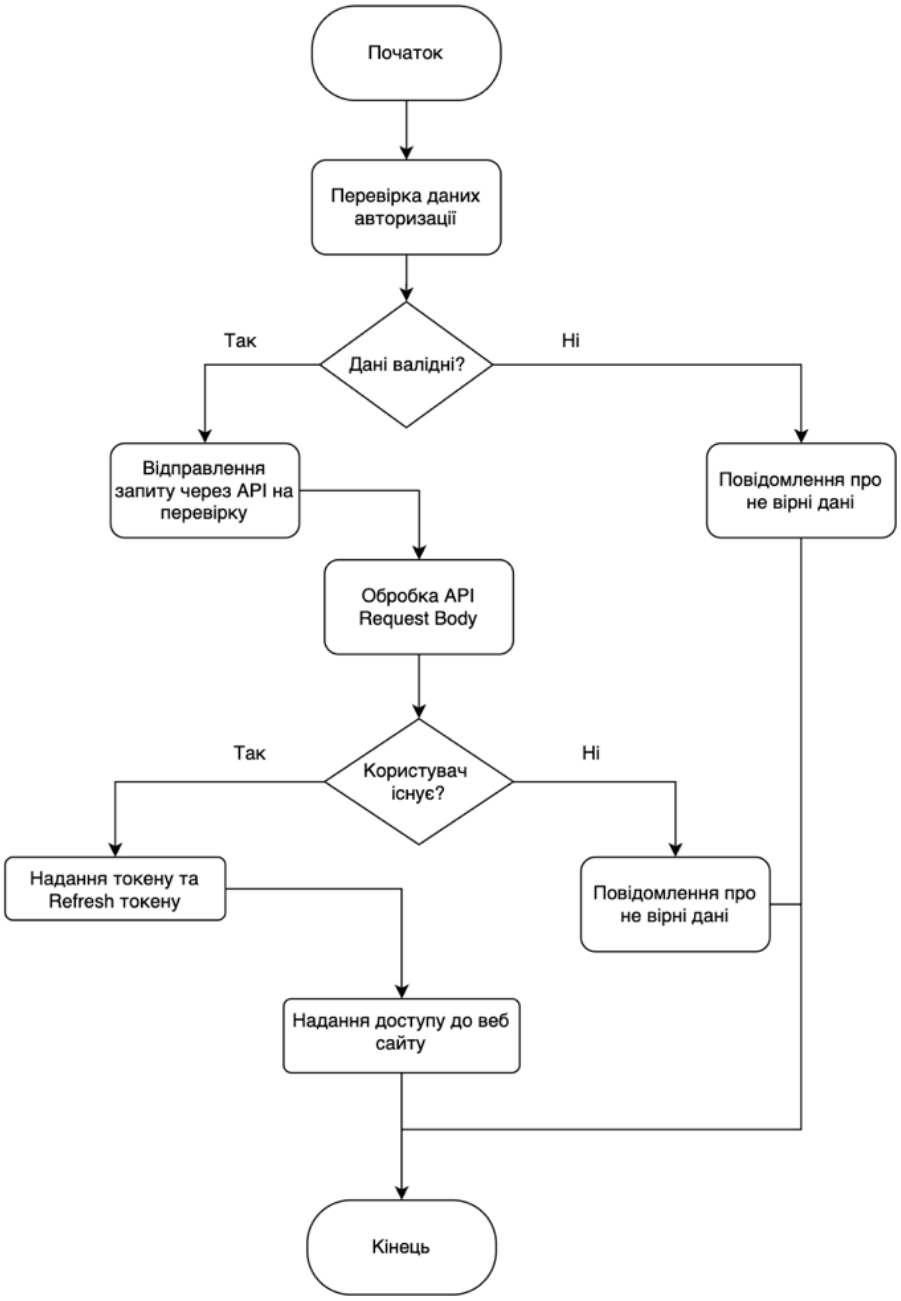


Рисунок 2.4 – Діаграма активності процесу авторизації користувача

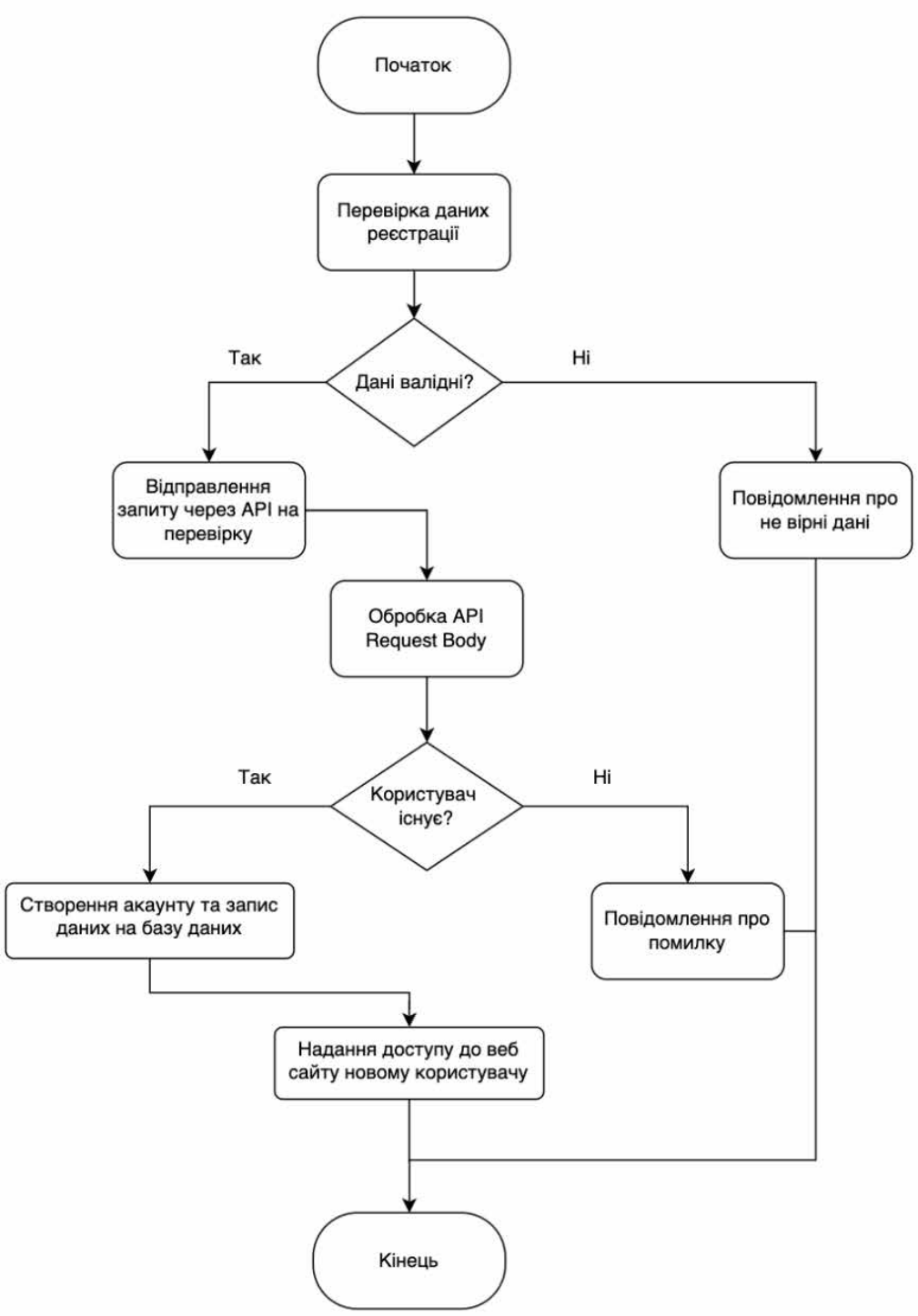


Рисунок 2.5 – Діаграма активності процесу реєстрації користувача

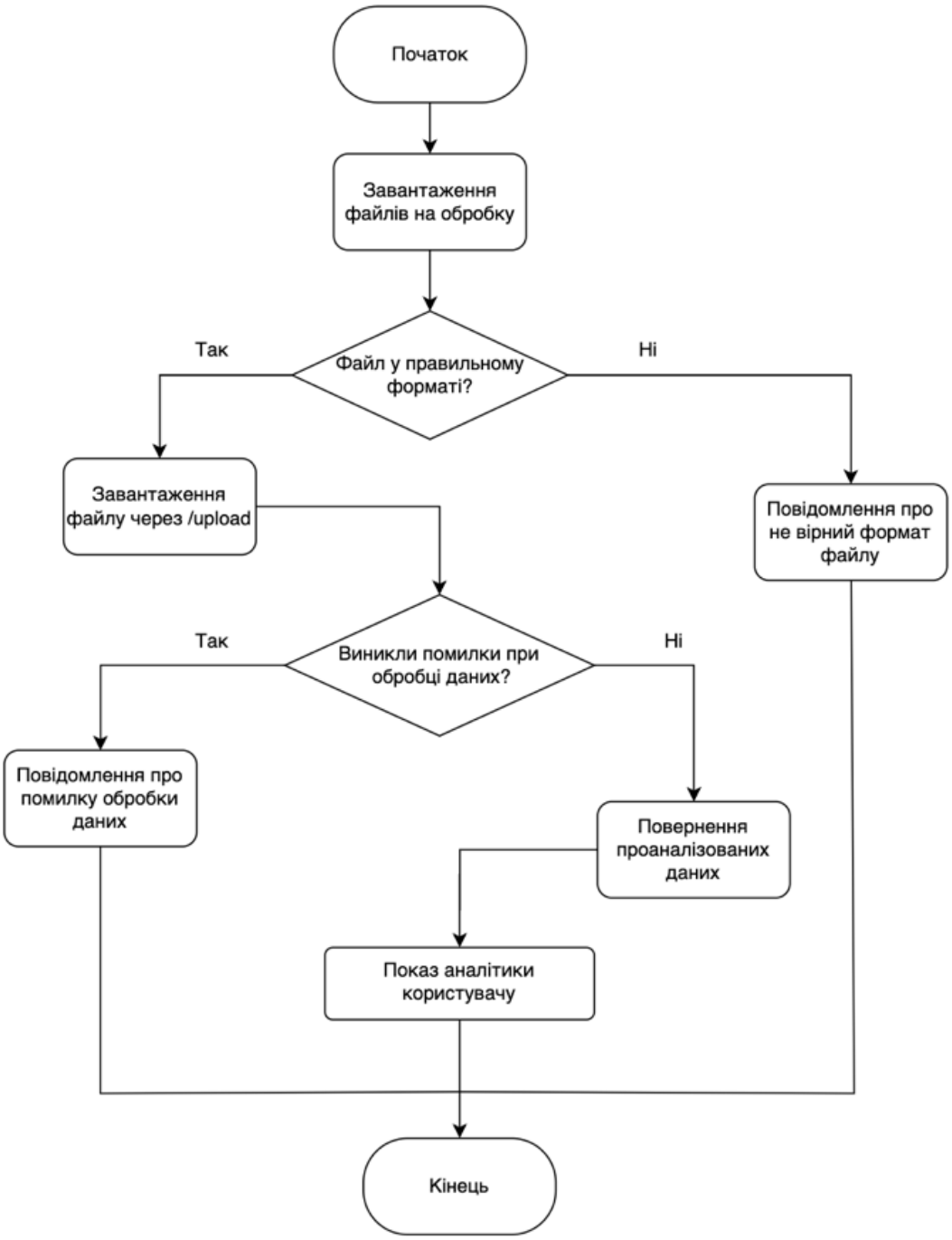


Рисунок 2.6 – Діаграма активності процесу завантаження та аналізу файлів

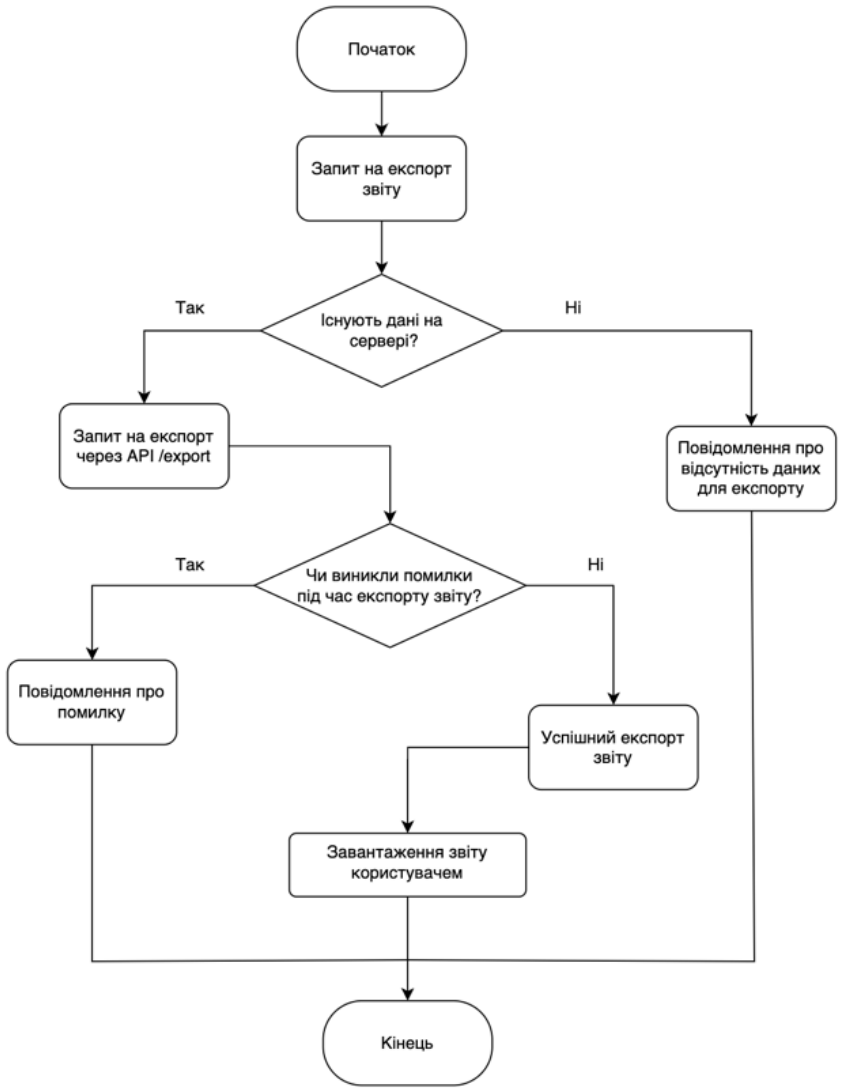


Рисунок 2.7 – Діаграма активності процесу експорту звіту

2.4 Інтерфейс програмування додатку

Інтерфейс програмування додатку (API) забезпечує зв’язок між серверною частиною та клієнтською частиною розробленого програмного забезпечення. Його правильне проєктування впливає на надійність, швидкодію та простоту підтримки системи. В цьому підрозділі ми розглянемо структуру API системи, представлену за допомогою діаграм послідовності та компонентів, включаючи

детальний огляд реалізованих маршрутів, методів, технологій та способів обробки даних.

2.4.1 Діаграма компонентів та реалізованих маршрутів API

На основі FastAPI розроблено RESTful API, структурований у вигляді окремих компонентів (модулів) для підтримки чіткої архітектури, простоти обслуговування та подальшого масштабування (див. табл. 2.3). FastAPI - це сучасний, швидкий (високопродуктивний), вебфреймворк для створення API за допомогою Python, в основі якого лежить стандартна анотація типів Python [9]. REST — абревіатура від REpresentational State Transfer (передача репрезентативного стану).

RESTful API — це архітектура інтерфейсу прикладних програм (API), що використовує HTTP-запити для доступу до ресурсів та їхнього використання. Такими HTTP-запитами є GET, PUT, POST і DELETE. Вони дають змогу, відповідно, читати, змінювати, створювати й видаляти ресурси [1].

Обрана комбінація архітектури API та веб фреймворк гарно підходять для теми системи для аналізу даних користувачів онлайн бібліотеки, адже вони надають швидке та якісне рішення, яке можна швидко розробити та підтримувати.

Серверна частина системи розділена на декілька основних компонентів та файлів, що кожен виконує свій алгоритм:

- main.py

Це центральний компонент, що запускає сервер FastAPI, підключає маршрути, проміжне програмне забезпечення, сервіси авторизації та з'єднання з базою даних.

- routes.py

Визначає основні кінцеві точки HTTP для авторизації, управління користувачами, отримання інформації про активність бібліотеки, списку завантажених файлів та налаштувань системи.

- analysis_routes.py

Реалізує спеціалізовані кінцеві точки для завантаження файлів, їх перевірки, запуску процесу аналізу та створення фінальних звітів.

- auth.py

Забезпечує захист API за допомогою JSON Web Tokens, реалізує методи реєстрації, авторизації та контролю доступу.

- database.py

Містить у собі логіку взаємодії з базою даних PostgreSQL, використовуючи бібліотеку SQLAlchemy для ORM-запитів, що дозволяє швидко отримувати, зберігати та оновлювати інформацію.

ORM (Object-Relational Mapping) — це підхід у програмуванні, який дозволяє працювати з базами даних за допомогою об’єктів. Кожен клас у програмі відповідає таблиці в базі даних, а кожен об’єкт — конкретному рядку цієї таблиці. Завдяки цьому розробники можуть оперувати даними як із звичайними змінними, а не писати SQL-запити вручну.

library_data.py

Реалізує процеси завантаження та збереження файлів, а також попередню обробку інформації для подальшого аналізу.

- library-analysis.py

Обробляє завантажені файли, здійснює аналітичні операції, розраховує ключові метрики, створює діаграми та формує звіти у форматі JSON для подальшого експорту.

- file_upload.py

Модуль, що відповідає за прийом файлів через API, перевірку їх структури, формату та відповідності заданим вимогам.

Компоненти інтерфейсу користувача побудовані на основі React.js. React JS — це відкритий JavaScript-фреймворк, а точніше, бібліотекою JavaScript, яка використовується для розробки інтерфейсів користувача. Він був створений компанією Facebook і швидко набув популярності серед розробників з усього світу. React дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у React JS є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за показ певної частини користувацького інтерфейсу [18]. А взаємодія із сервером реалізована через чітко визначені кінцеві точки API (див. табл. 2.3):

- api.js

Основний компонент інтерфейс-сторони клієнту, який реалізує HTTP-запити до серверної частини, використовуючи метод axios, обробляє відповіді та помилки. Axios — це стороння бібліотека, яка також фокусується на роботі з HTTP-запитами, але має кілька своїх бонусів:

- Легкість коду: Коду з Axios часто менше завдяки більшій кількості інструментів для настройки.
- Автоматична обробка JSON: Axios автоматично перетворює JSON-дані, що знижує необхідність у ручній обробці [10].

Таблиця 2.3

Основні методи API

№	**Метод**	**Маршрут**	**Опис**
1	2	3	4
1	POST	/api/auth/login	Авторизація користувача
2	POST	/api/auth/register	Реєстрація нового аналітика
3	GET	/api/files	

			Отримання списку завантажених файлів
4	POST	/api/files/upload	Завантаження нового файлу даних

Кінець таблиці 2.3

1	2	3	4
5	DELETE	/api/files/{file_id}	Видалення завантаженого файлу
6	POST	/api/analysis	Ініціювання аналізу завантажених файлів
7	GET	/api/analysis/reports	Отримання списку звітів аналізу
8	POST	/api/reports/export	Експорт звіту в PDF, CSV, JSON, XLSX

Джерело: створено автором на основі [9]

2.4.2 Технології, використані при реалізації API

У таблиці 2.4 наведено список інструментів та технологій, використаних у серверній частині системи, а також їхні ролі у побудові стабільного API.

Таблиця 2.4

Список використаних технологій

№	**Технологія**	**Опис та переваги**
1	2	3
1	FastAPI	Сучасний асинхронний фреймворк для побудови REST API на Python 3.6+. - Підтримка для асинхронної обробки запитів. - Автоматична генерація документації (Swagger UI) [15]. - Повна підтримка типізації завдяки інтеграції з Pydantic [26].

		- Висока продуктивність.

Кінець таблиці 2.4

1	2	3
2	JWT	Механізм безпечної авторизації [11]. - Токен передається в заголовку Authorization, не потребує збереження сесій на сервері. - Декодування дозволяє контролювати доступ на основі вмісту токена (роль, ID тощо).
3	SQLAlchemy	ORM для Python, що абстрагує роботу з PostgreSQL [22]. - Безпечне формування запитів. - Підтримка транзакцій. - Генерація схем бази даних із моделей. - Захист від SQL-ін’єкцій.
4	Pydantic	Бібліотека перевірки даних, що інтегрується з FastAPI [26]. - Автоматична генерація JSON-схем для документації. - Серіалізація вихідних моделей.
5	Uvicorn	Легкий сервер, який запускає FastAPI-додаток. • Підтримка багато потокової обробки запитів. • Сумісність з HTTP/2, WebSocket та asyncio. • Мінімальне споживання ресурсів і висока продуктивність [23].

Джерело: створено автором на основі [11][15][22][23][26]

2.4.3 Обробка помилок

Для того щоб реалізувати стабільну та контрольовану роботу API, реалізовано глобальний обробник помилок, який покриває типові статуси [1] [9]:

- 400 Bad Request — неправильний формат вхідних даних (наприклад, не правильний формат JSON або відсутні обов’язкові поля).
- 401 Unauthorized — невірний токен авторизації або його відсутність.
- 404 Not Found — спроба звернення до неіснуючого ресурсу (наприклад, файл або звіт не знайдено).

- 500 Internal Server Error — помилки на стороні сервера, що не передбачені в бізнес-логіці.

Усі помилки повертаються у структурованому форматі JSON з полем, що містить опис проблеми для користувача або розробника.

Також для контролю роботи API в системі реалізовано запис помилок за допомогою виведення помилок у консоль:

- час кожного запиту;
- IP-адреса клієнта;
- маршрут запиту;
- відповідь API або повідомлення про помилку;
- статус код відповіді.

Виведення помилок у консолі дозволяє вести аудит роботи системи, виявляти критичні точки і швидко діагностувати несправності. Створена інфраструктура забезпечує гнучкий, безпечний і зручний для розширення API, що повністю відповідає сучасним стандартам розробки веб-сервісів [20].

2.4.4 Діаграма компонентів

Для кращого розуміння архітектури створено діаграму компонентів (див. рис. 2.8), яка ілюструє структуру розробленого програмного забезпечення. Вона розділена на дві частини: серверну та клієнтську, які взаємодіють між собою через API.

У серверній частині, що розміщується в Docker-контейнері, розміщено всі основні файли серверної частини на Python з використанням FastAPI: обробники маршрутів, логіка авторизації, обробка файлів і даних, аналітика та звітність.

Docker — це інструмент з відкритим вихідним кодом, який забезпечує ізольоване середовище для запуску програм у вигляді контейнерів. Його використання дозволяє ефективно розподіляти ресурси системи, пришвидшити

розгортання програмного забезпечення та забезпечити стабільну роботу застосунків при перенесенні між різними середовищами, включаючи локальні машини та хмарні платформи [8].

Клієнтська частина побудована на React.js [19] і складається з компонентів інтерфейсу: форми логіну та реєстрації, завантаження файлів, виведення статистики, візуалізації результатів аналізу, експорт звітів. Центральним є файл App.jsx, який поєднує всі інші елементи у цілісний інтерфейс.

Розглянемо максимально детально основні компоненти системи розглянуто у таблицях 2.5 та 2.6, де відповідно таблиця 2.6 це компоненти серверної частини, а таблиця 2.6 – інтерфейсної частини. У таблиці 2.7 детально розглянуто структуру файлів Docker та розгортання системи.

Таблиця 2.5

Компоненти серверної частини

№	Файл / Модуль	Призначення
1	2	3
1	main.py	Запускає сервер FastAPI, підключає маршрути та компоненти.
2	routes.py. analysis_routes.py	Містять визначення кінцевих точнок REST API: завантаження файлів, аналіз, створення звітів.
3	auth.py	Відповідає за авторизацію користувачів (аналітиків), реалізація JWT.
4	analyst.py	Керує інформацією про користувачів, їхні ролі та права.
5	database.py	Підключення до PostgreSQL, управління моделями та запитами.
6	library_data.py, library_analysis.py	Аналіз даних, розрахунок метрик, підготовка звітів.
7	file_upload.py	Обробка завантаження, збереження та перевірки файлів даних.

Джерело: створено автором

Таблиця 2.6

Компоненти інтерфейсної частини

№	Файл / Компонент	Призначення
1	2	3
1	App.jsx	Головний компонент програми, ініціалізація маршрутизації та глобальних стилів.
2	DashboardLayout.jsx	Основний шаблон інтерфейсу з навігаційною панеллю.
3	UploadFile.jsx	Завантаження файлів користувачем, відображення статусу.
4	Analytics.jsx + дочірні: AnalyticsFilter.jsx, ActivityChart.jsx, GenreChart.jsx, BooksTable.jsx	Візуалізація статистичних даних (графіки, таблиці), фільтрація.
5	ExportReport.jsx	Експорт готових звітів (PDF, CSV, JSON).
6	AuthPage.jsx, LoginForm.jsx, RegisterForm.jsx	Сторінки входу/реєстрації, обробка логіну.
7	Settings.jsx	Зміна налаштувань профілю користувача, мови та інших параметрів.

Джерело: створено автором

Таблиця 2.7

Інфраструктура розгортання Docker контейнера

№	Файл / Компонент	Призначення
1	Dockerfile	Інструкція для створення образу серверної частини.
2	Docker-compose.yml	Описує запуск усіх контейнерів (FastAPI, PostgreSQL тощо).

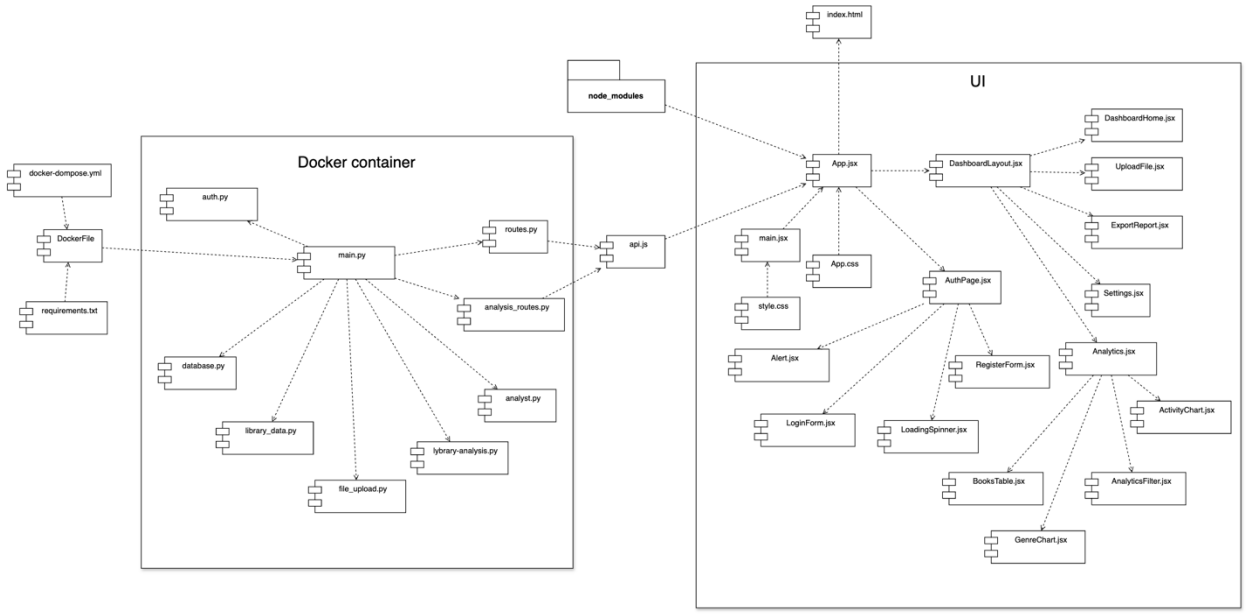
3	Requirements.txt	Містить список Python-бібліотек, необхідних для роботи бекенду.

Джерело: сформовано автором

- Взаємодія між компонентами (API):

Компоненти взаємодіють через REST API, який надає стандартизований інтерфейс для передачі даних у форматі JSON між користувачською стороною та серверною частиною. Запити виконуються через HTTP-протокол з використанням методів GET, POST, PUT та DELETE (див. табл. 2.3) [9] [20].

Рисунок 2.8 – Діаграма компонентів розроблюваної системи



2.5 Безпека та тестування

Захист персональних даних, забезпечення контрольованого доступу до системи та перевірка стабільності функціоналу є максимально важливим аспектам при створенні програмного забезпечення для обробки аналітичних даних. У процесі розробки даної системи для аналізу активності користувачів

онлайн-бібліотеки були реалізовані сучасні практики безпеки та тестування, що відповідають загальновизнаним стандартам [5].

2.5.1 Реалізація безпеки

Система включає комплексний підхід до забезпечення інформаційної безпеки, враховуючи рекомендації Web content accessibility guidelines (WCAG) [25].

A. Аутентифікація та контроль доступу:

Усі критичні кінцеві точки захищені JWT-автентифікацією. Після входу в систему користувач отримує токен, який передається в заголовках запитів до API:

- Токен має обмежений час дії.
- Токен перевіряється при кожному запиті на захищений маршрут через проміжне ПЗ.
- У разі недійсного або відсутнього токена повертається HTTP 401.

B. Захист облікових даних

Паролі зберігаються у зашифрованому вигляді:

- Під час реєстрації пароль шифрується.

Цей підхід виключає можливість зчитування паролів навіть при проникненні небажаного актору до бази даних.

C. Перевірка та обробка вхідних даних

Основним механізмом валідації є використання моделей Pydantic, які забезпечують автоматичну перевірку структури тіла запиту відповідно до визначених правил [26]. Моделі створюються як класи, що успадковуються від BaseModel, із чітко заданими типами атрибутів. Це дозволяє:

- Автоматично перевіряти типи і значення вхідних параметрів;
- гарантувати, що всі обов’язкові поля присутні та мають коректний формат;
- серіалізувати та десеріалізувати дані відповідно до очікуваної структури;

- формувати специфікацію OpenAPI без додаткових зусиль.
- Усі необов’язкові поля мають стандартизовані значення.

D. Обмеження доступу до файлів

- Усі файли даних та звіти прив’язані до конкретного користувача (аналітика). При запиті ресурсу виконується перевірка, чи збігається ID користувача, що виконує запит, з власником ресурсу.

E. Захист від CSRF/XSS

Cross-Site Request Forgery (CSRF), або міжсайтове підроблення запиту, — це тип кібератаки, при якому зловмисник змушує користувача, що вже пройшов автентифікацію на певному сайті, несвідомо виконати небажану дію на цьому сайті. Така атака може відбутися, якщо користувач, не виходячи з облікового запису, переходить на шкідливу сторінку або відкриває небезпечне посилання. У результаті сервер може прийняти запит як законний, наприклад, змінити дані профілю або здійснити переказ коштів.Оскільки інтерфейс системи працює як SPA (Single Page Application) із використанням React.js [19], всі взаємодії з сервером йдуть через API. Це дозволяє уникати CSRF, оскільки інтерфейс не містить функцій вбудовування HTML з неперевіреними даними.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Підготовка середовища розробки

На старті розробки програмного забезпечення одним із пріоритетів стало створення стабільного, ізольованого та уніфікованого середовища, яке мінімізує ризики технічної несумісності бібліотек, залежностей, інструментів і версій. Щоб забезпечити ефективну реалізацію системи аналізу активності користувачів онлайн-бібліотеки, було впроваджено сучасні компоненти, які спрощують розробку як серверної, так і клієнтської частини — швидко, якісно та зручно. Така основа не лише усуває потенційні конфлікти сумісності, а й створює сприятливі умови для масштабування, тестування і подальшого розгортання системи, завдяки використанню продуктивних, гнучких інструментів з активною підтримкою спільноти розробників.

3.1.1 Обґрунтування вибору інструментів та середовища

Для створення програмного забезпечення був обраний редактор коду Visual Studio Code (VS Code) [28], який характеризується широкими можливостями, зручністю та універсальністю. VS Code забезпечує підтримку різних мов програмування (Python, JavaScript, HTML, CSS), має потужну систему плагінів, які значно спрощують і прискорюють розробку. На відміну від інших редакторів, таких як PyCharm [29], VS Code більш універсальний, підтримує роботу з кількома мовами та інтегрує всі необхідні функції розробки (наприклад, інтеграція з Git).

Для контролю версій коду був використаний Git, що дозволяє відстежувати історію змін та працювати в команді. У якості платформи для зберігання коду та спільної роботи використовувався сервіс GitHub [27], що забезпечує зручний і

швидкий доступ до проєкту з будь-якого місця та сприяє ефективній командній роботі (див. рис. 3.1).

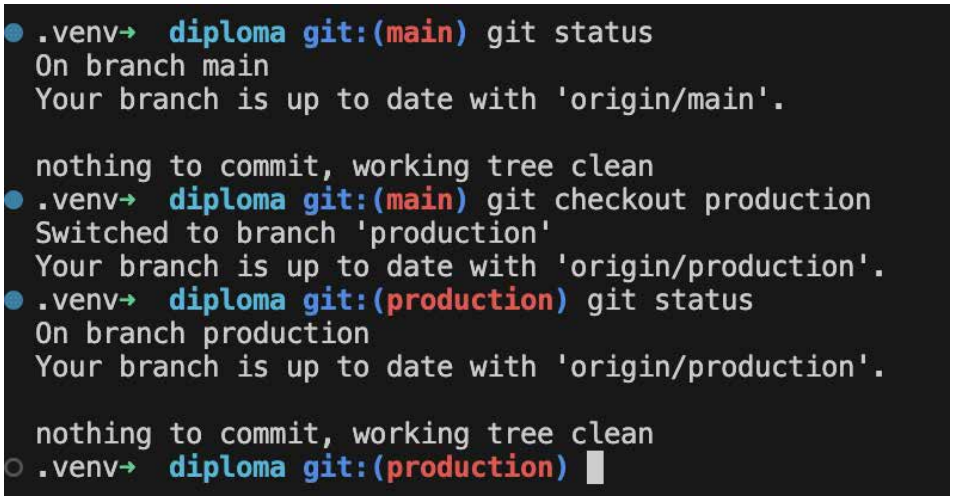


Рисунок 3.1 — Приклад використання Git у редакторі VS Code.

3.1.2 Встановлення та налаштування технологій розробки

Для організації серверного середовища було встановлено мову програмування Python 3.11, після чого створено віртуальне середовище командою: *python -m venv venv*

Після активації середовища усі необхідні залежності було встановлено з файлу *requirements.txt*, який містив бібліотеки FastAPI, Pandas, SQLAlchemy, Uvicorn, Pydantic та інші, зміст файлу міститься на рисунку 3.2. Повний перелік основних технологій програмного забезпечення зазначено у таблиці 3.1.

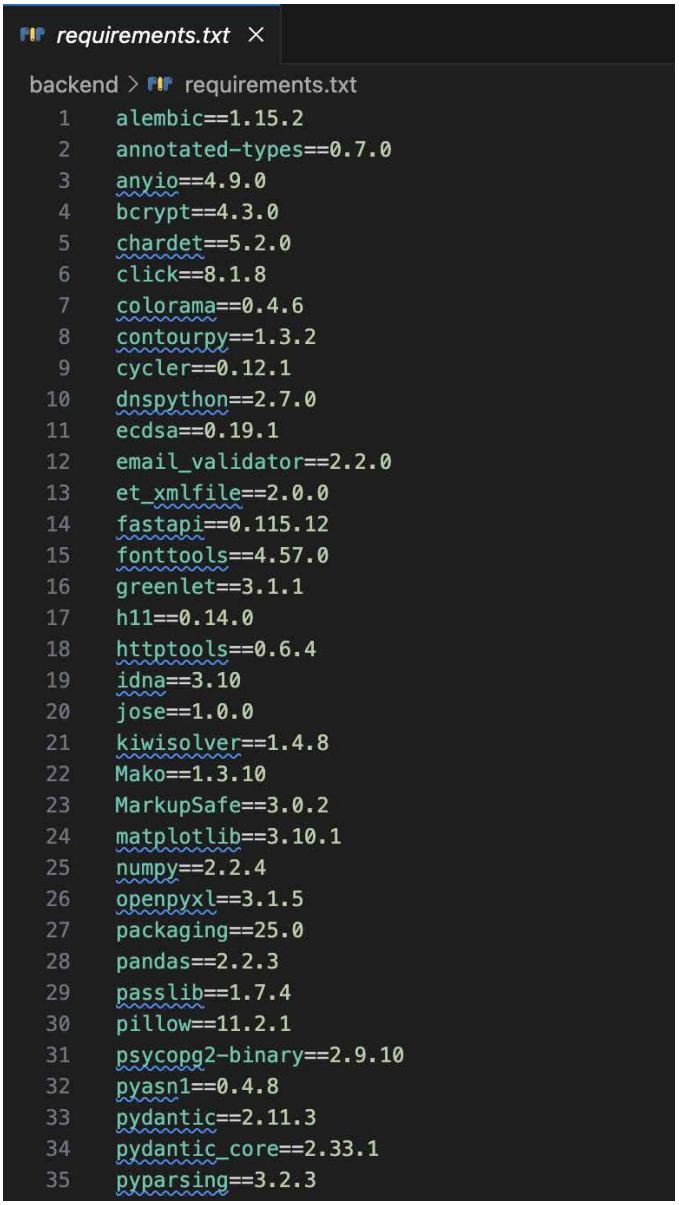


Рисунок 3.2 – Зміст файлу з залежностями requirements.txt

Паралельно було розгорнуто середовище для клієнтської частини. Встановлено Node.js [30], після чого створено проєкт за допомогою Vite [31] (рисунок 3.3).

Рисунок 3.3 – Ініціалізація проєкту за допомогою Vite

Для реалізації інтерфейсу використовувались бібліотеки React.js, Axios, React Router, Chart.js. Їх було встановлено окремо (рис. 3.4).

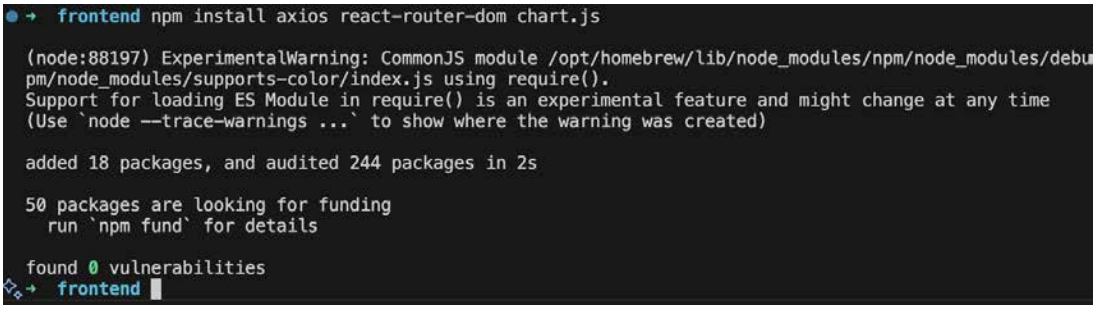


Рисунок 3.4 – Встановлення додаткових бібліотек у терміналі

Усі налаштування середовища зберігалися у відповідних файлах, які містять секретні змінні (ключі доступу, URL до бази даних, токени). Для керування всіма компонентами застосовано Docker [8]. Створено файл Dockerfile для серверної частини та docker-compose.yml, який поєднує FastAPI, PostgreSQL та фронтенд у єдиний комплект. Запуск здійснювався командою: *docker-compose up –build.* На рисунку 3.5 зображено процес запуску Docker контейнеру.

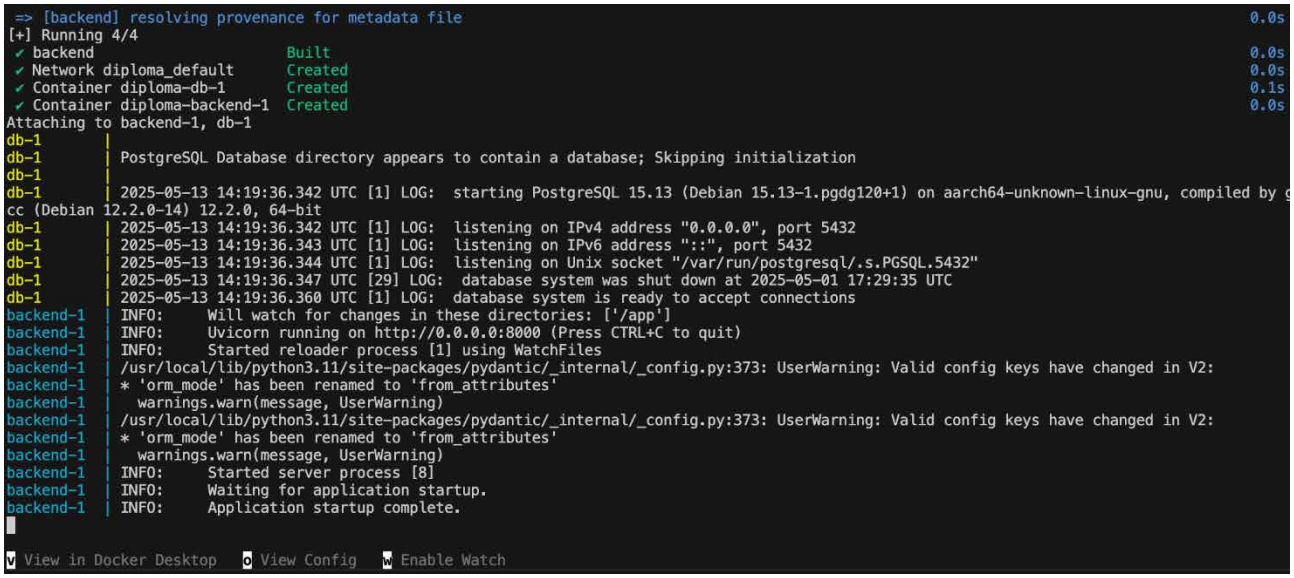


Рисунок 3.5 – Запуск Docker контейнеру для локальної розробки

Таблиця 3.1

Перелік використаних технологій ПЗ

№	Компонент	Технологія	Версія
1	2	3	4
1	Серверна частина (backend)	Python, FastAPI	3.11, 0.110
2	База даних	PostgreSQL	15.2
3	Клієнтська частина	React.js, Vite	18+, 4+
4	Контейнеризація	Docker, Docker Compose	24+, 1.29+
5	Редактор коду	Visual Studio Code	1.86+
6	Хостинг сервіс	Render.com, Vercel.com	-

Джерело: сформовано автором

3.2 Реалізація серверної частини

3.2.1 Структура серверної частини

Код серверної частини має структуровану організацію та складається з кількох модулів, кожен з яких відповідає за окремі задачі, структура файлової системи зображення на рис. 3.6.

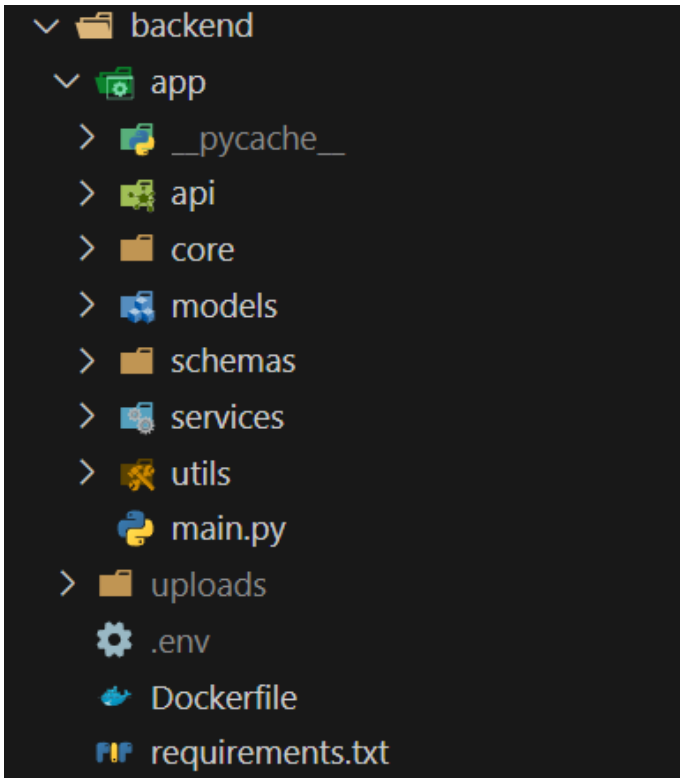


Рисунок 3.6 – Структура файлів серверної частини.

3.2.2 Реалізація авторизації в системі

Для захисту даних та забезпечення безпечного доступу до функцій системи реалізовано механізм авторизації на основі JWT-токенів.

Алгоритм авторизації працює наступним чином:

А. Користувач вводить свій логін (електронну пошту) та пароль у форму авторизації.

B. Система отримує ці дані, перевіряє їх на правильність, порівнюючи введені дані з інформацією в базі даних.
C. Якщо дані коректні, сервер створює унікальний JWT-токен, в якому зашифрована інформація про користувача та термін дії доступу.
D. Цей токен повертається користувачу та надалі використовується для всіх наступних запитів до системи.

Цей підхід дозволяє забезпечити високий рівень безпеки, оскільки токен перевіряється при кожному запиті, а паролі користувачів зберігаються у базі даних у захищеному, зашифрованому вигляді, повний відрізок функціональності авторизаціїї у системі зазначено у Додатку Б.

3.2.3 Завантаження, збереження та обробка файлів користувачів

У системі реалізована можливість завантаження файлів, які містять дані для аналізу (у форматі JSON). Після того, як користувач завантажує файл, система автоматично виконує низку перевірок (див. відрізок коду 2):

A. Перевіряє коректність формату та структури файлу (наявність необхідних полів та даних).
B. Перевіряє відсутність дублікатів, помилок чи пропусків у даних.
C. Після перевірки файл зберігається у спеціально визначеному каталозі або у базі даних разом із інформацією про те, хто саме завантажив ці дані.
D. Алгоритм обробки завантаженого файлу:
E. `Користувач завантажує файл → сервер перевіряє файл → у разі успіху зберігає дані → запускає автоматичну обробку для формування статистики.`

Повний відрізок коду зазначено у Додатку. Б

3.2.4 Формування статистичних звітів

Процес формування статистичних звітів проходить через декілька етапів:

A. Завантаження та попередня підготовка даних

Спочатку завантажені користувачем файли у форматі JSON зчитуються методом load_data, який витягує інформацію про користувачів бібліотеки та їхню активність. Потім, метод initialize_dataframes розбиває ці дані на окремі табличні представлення (DataFrame-структури), такі як:

- Основні дані користувачів (реєстрація, особисті дані).
- Інформація про переглянуті та взяті книги.
- Інформація про тривалість та частоту відвідувань онлайн-бібліотеки.
- Історія пошукових запитів .

B. Обробка та перетворення даних

Наступним кроком дані піддаються додатковій обробці, де, зокрема, метод process_datetime_columns перетворює рядки з датами у спеціальні формати, що дозволяє легко виконувати пошук і аналіз за будь-яким часовим проміжком .

C. Безпосередній аналіз даних

Після попередньої обробки система виконує серію модулів аналізу, кожен з яких відповідає за конкретну метрику:

Модуль аналізу активності (analyze_usage_patterns): визначає активність користувачів за періодами (годинами, днями тижня, місяцями) та середню тривалість перебування у бібліотеці.

Модуль аналізу популярності контенту (analyze_content_performance): визначає найбільш популярні книги, категорії та жанри на основі частоти переглядів та оцінок користувачів.

Модуль сегментації користувачів (analyze_user_segments): виконує розподіл користувачів за різними критеріями, такими як вік, рівень освіти, тип акаунту тощо.

Модуль аналізу пошукових запитів (analyze_search_patterns): аналізує ключові слова, які використовують читачі, та визначає частоту і час пошукових запитів.

Модуль аналізу утримання користувачів (analyze_retention): оцінює, як часто користувачі повертаються до бібліотеки після першого відвідування .

D. Формування узагальненого звіту

Останнім етапом процесу є метод generate_comprehensive_report (див. відрізок коду 3), який агрегує результати аналізу у загальну структуру (у вигляді словника чи JSON-файлу), яка містить усі обчислені метрики. Отриманий звіт зберігається в базі даних PostgreSQL та може бути наданий користувачу у вигляді фінального документа .

a. Алгоритм формування звітів крок за кроком:
b. Користувач завантажує файл із даними.
c. Сервер перевіряє коректність і зберігає файл.
d. Дані перетворюються у формат, зручний для аналізу (DataFrame).
e. Виконуються аналітичні модулі (активність, популярність, сегментація).
f. Система формує узагальнений звіт.
g. Результати аналізу зберігаються у базу даних.

Користувач отримує можливість завантажити звіт у форматі PDF, CSV або JSON через клієнтську частину інтерфейсу.

У Додатку В можете ознайомитись з повною кодовою версією логіки формування статистичного звіту

Отже, сконструйована серверна частина системи охоплює весь цикл роботи з даними — від їх завантаження та аналізу до безпечного зберігання й генерації аналітичних звітів у зручному для користувача форматі. Завдяки сучасним технологічним рішенням вдалося створити швидкий, надійний і безпечний серверний модуль, що легко інтегрується з клієнтським інтерфейсом і відповідає потребам кінцевих користувачів.

3.3 Реалізація клієнтської частини

Клієнтська частина програмного забезпечення була реалізована з використанням бібліотеки React, яка дозволяє створювати динамічні, зручні та масштабовані інтерфейси користувача. Основна мета клієнтської частини — забезпечити аналітику комфортним візуальним середовищем для взаємодії з системою: завантаженням файлів, переглядом результатів аналізу, формуванням і експортом звітів.

Інтерфейс побудовано за принципами односторінкового застосунку (Single Page Application, SPA) — сучасного підходу до веб-розробки, при якому вся робота програми відбувається в межах однієї веб-сторінки. На відміну від традиційних багатосторінкових сайтів, де кожна дія користувача вимагає повного перезавантаження сторінки з сервера, SPA працює інакше:

- Після першого завантаження всі необхідні ресурси (HTML, CSS, JavaScript) завантажуються одноразово
- Подальша взаємодія відбувається динамічно через API – програма лише довантажує необхідні дані
- Перехід між розділами відбувається миттєво, без видимих перезавантажень

Такий підхід особливо ефективний для аналітичних систем, де користувач постійно працює з різними видами даних і потребує швидкого перемикання між інструментами аналізу.

3.3.1 Структура компонентів

Архітектура інтерфейсу має чіткий поділ на функціональні компоненти (див. рисунок 3.7 та табл.2.6 у розділі 2)

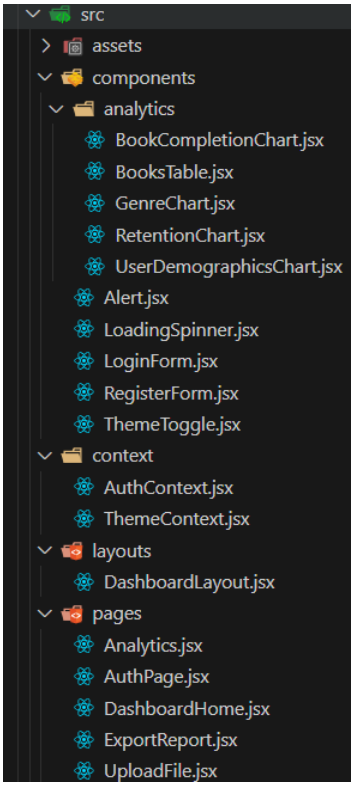


Рисунок 3.7 — Структура React-компонентів у файловій системі проєкту.

3.3.2 Реалізація ключових функцій інтерфейсу

Авторизація та реєстрація користувача реалізовані через форми, які надсилають POST-запити на сервер. У відповідь система повертає JWT-токен, який зберігається у локальному сховищі браузера. Після цього користувач отримує доступ до основного інтерфейсу аналітики.

“Дашборд” — це головна панель аналітика. Вона включає в себе такі компоненти (див. рис. 3.8):

- Панель навігації (зліва)
- Візуалізацію даних (графіки, діаграми, таблиці)

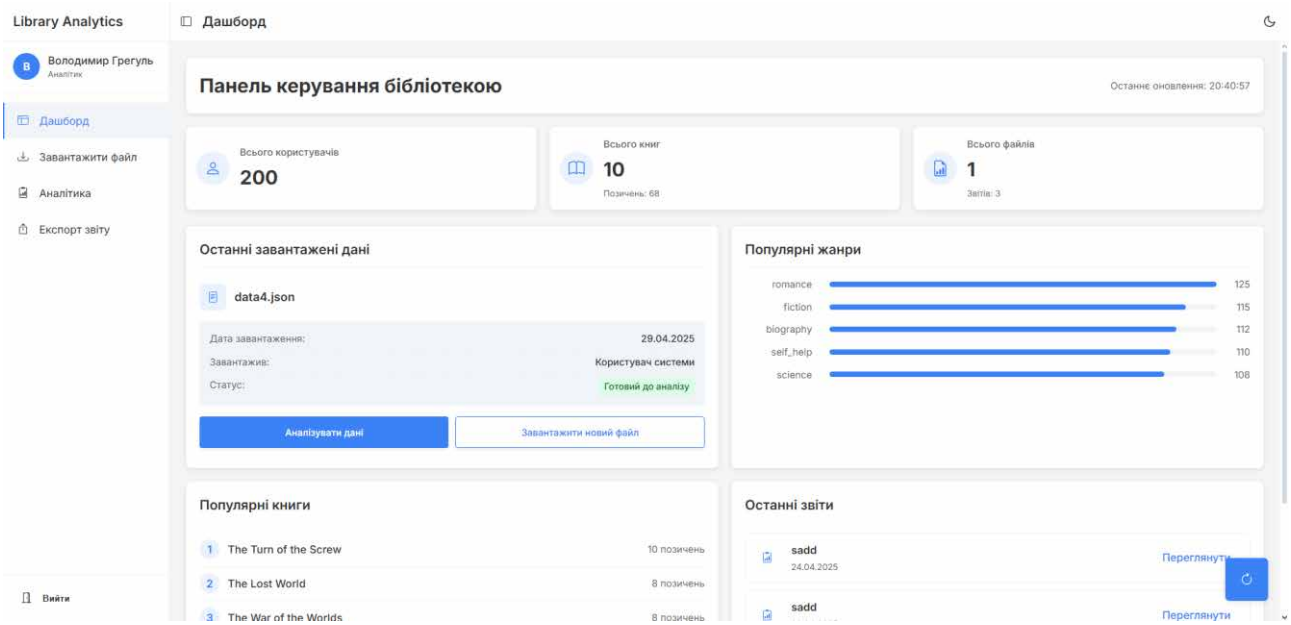


Рисунок 3.8 – Головна сторінка, Dashboard.jsx

Завантаження файлів відбувається через компонент UploadFile.jsx, який дозволяє користувачеві перетягнути файл у вікно або обрати через діалог. Після натискання кнопки "Завантажити", файл надсилається на сервер через API, а система повідомляє про успіх чи помилки (див. рис 3.9).

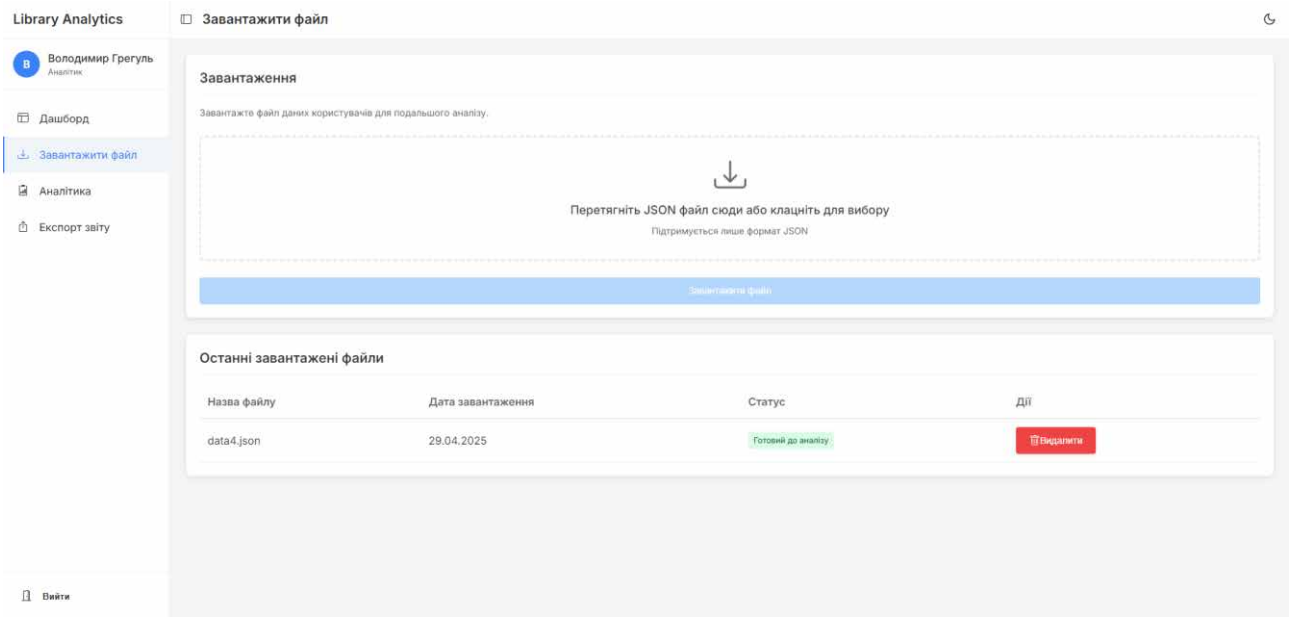


Рисунок 3.9 – Сторінка завантаження, UploadFile.jsx

Аналітика формується в компоненті Analytics.jsx. Після завантаження даних відбувається їх аналіз (на сервері), а у відповідь користувач отримує підготовлені структури, які відображаються у вигляді (див. рис. 3.10):

- Кругових діаграм популярних жанрів
- Лінійних графіків активності по днях
- Таблиць з метриками (кількість активних користувачів, тривалість сесій тощо)

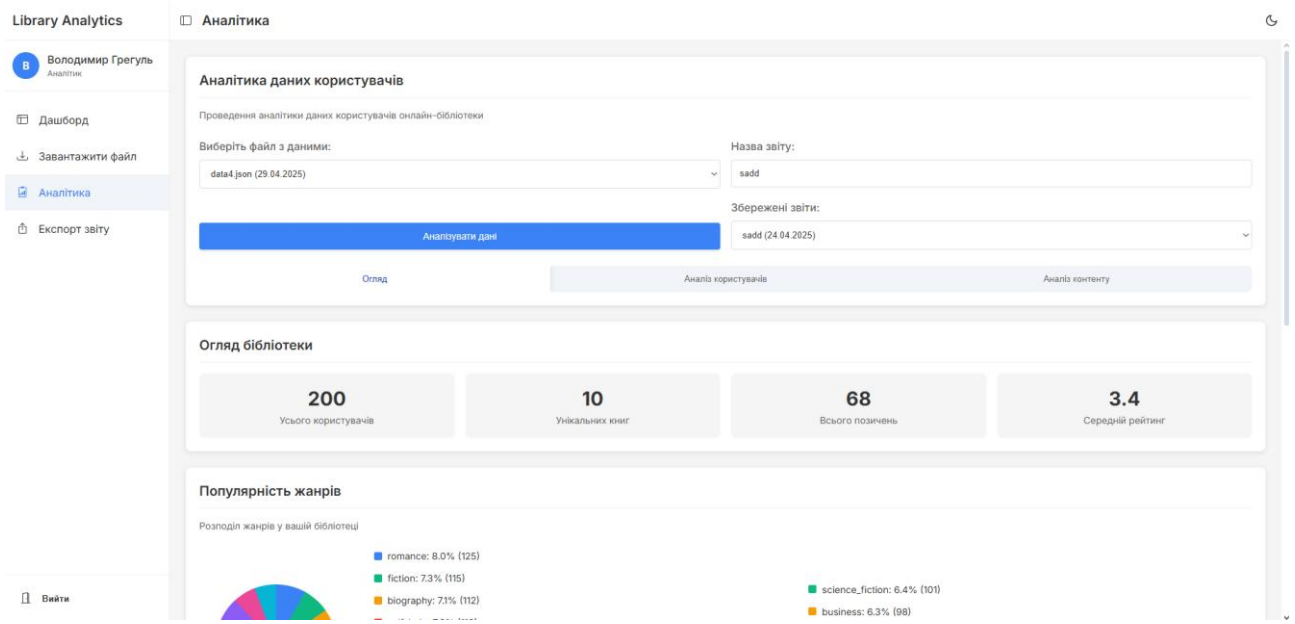


Рисунок 3.10 – Сторінка аналітики, Analytics.jsx

Експорт звіту реалізовано через компонент ExportReport.jsx, де користувач може обрати формат (PDF, CSV або JSON) та натиснути кнопку "Завантажити звіт" (див. рис. 3.11).

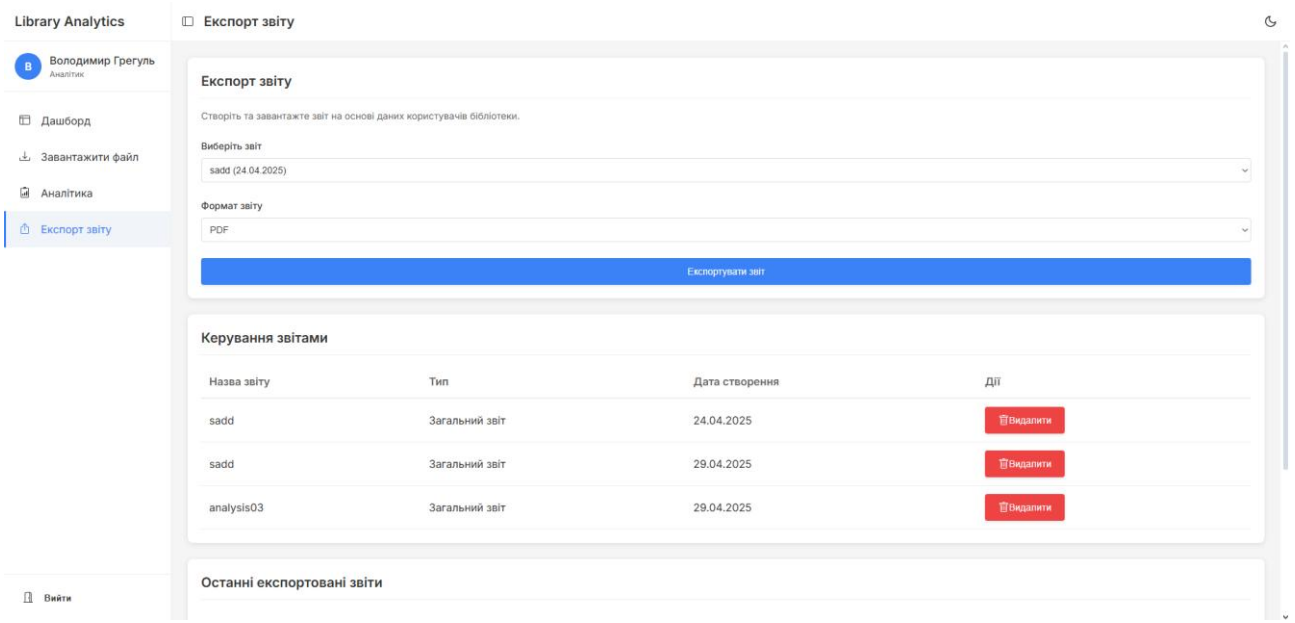


Рисунок 3.11 – Експорт звіту, ExportReport.jsx.

3.4 Документація та тестування API

Однією з переваг використання FastAPI є вбудована інтерактивна документація API, яка автоматично генерується на основі опису маршрутів та моделей валідації даних. Це забезпечує зручний інструмент для розробників і аналітиків, які можуть перевірити роботу кожного маршруту без використання сторонніх засобів.

Система надає дві основні форми документації:

- Swagger UI — графічний інтерфейс, у якому можна тестувати API безпосередньо в браузері [15].
- ReDoc — текстово-структурована версія для зручного читання API-інтерфейсів [15].

Ця документація створюється автоматично завдяки специфікації OpenAPI 3.0 [15], яку FastAPI підтримує з коробки (див. рис. 3.12). Основні маршрути API позначено у таблиці 6 [15].

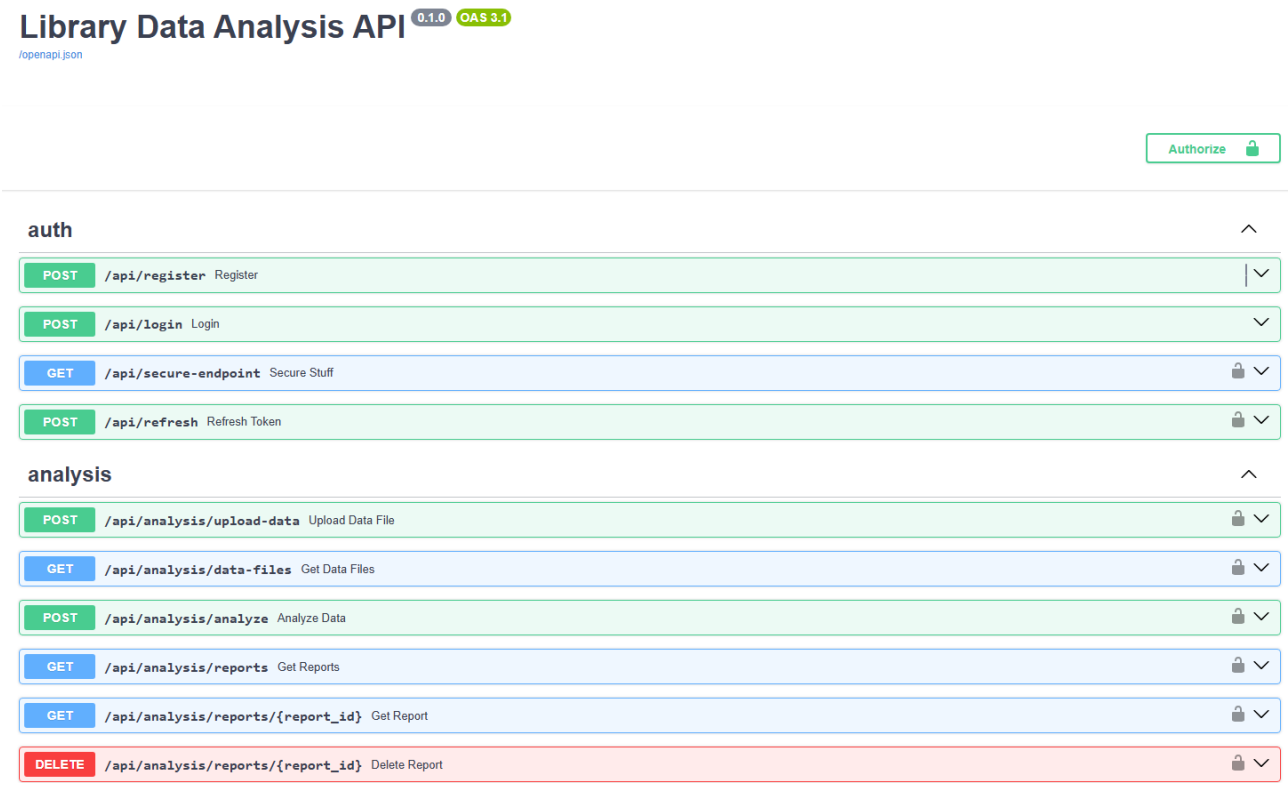


Рисунок 3.12 — Інтерфейс Swagger-документації FastAPI.

3.4.1 Тестування через Postman

Для перевірки коректності роботи програмного інтерфейсу (API) на етапі розробки та налагодження використовувався інструмент Postman [17] — одна з найпопулярніших утиліт для тестування REST API. Цей інструмент дозволяє вручну або автоматично надсилати HTTP-запити до сервера, аналізувати отримані відповіді та перевіряти поведінку системи у різних ситуаціях.

Зокрема, у Postman було створено окреме робоче середовище з набором колекцій, які охоплюють всі ключові маршрути API системи.

Кожен запит в колекції має налаштовану схему авторизації через Bearer Token, що дає змогу легко тестувати захищені частини API, ідентично до роботи клієнтської частини (див. рис. 3.13).

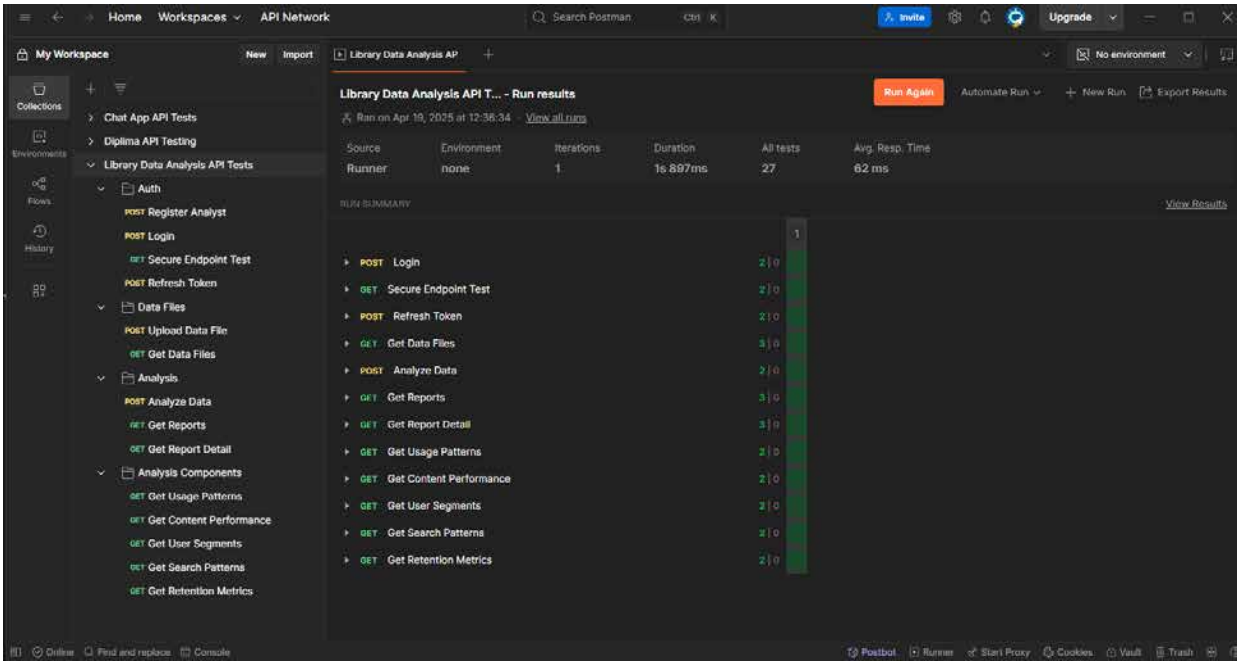


Рисунок 3.13 – Тестування захищеного API-запиту через Postman

3.5 Розгортання системи

Для розгортання серверної частини була вибрана платформа Render [7] (див. рис. 3.14), яка дозволяє легко розгортати Docker-контейнери без додаткових налаштувань. Для клієнтської частини обрано сервіс Vercel [24] (див рис. 3.15), який спеціалізується на розгортанні веб-інтерфейсів, забезпечуючи високу швидкість завантаження та автоматичні оновлення.

Таким чином, вибрані технології та інструменти дозволили створити надійне, швидке, зручне та масштабоване середовище для розробки програмного забезпечення.

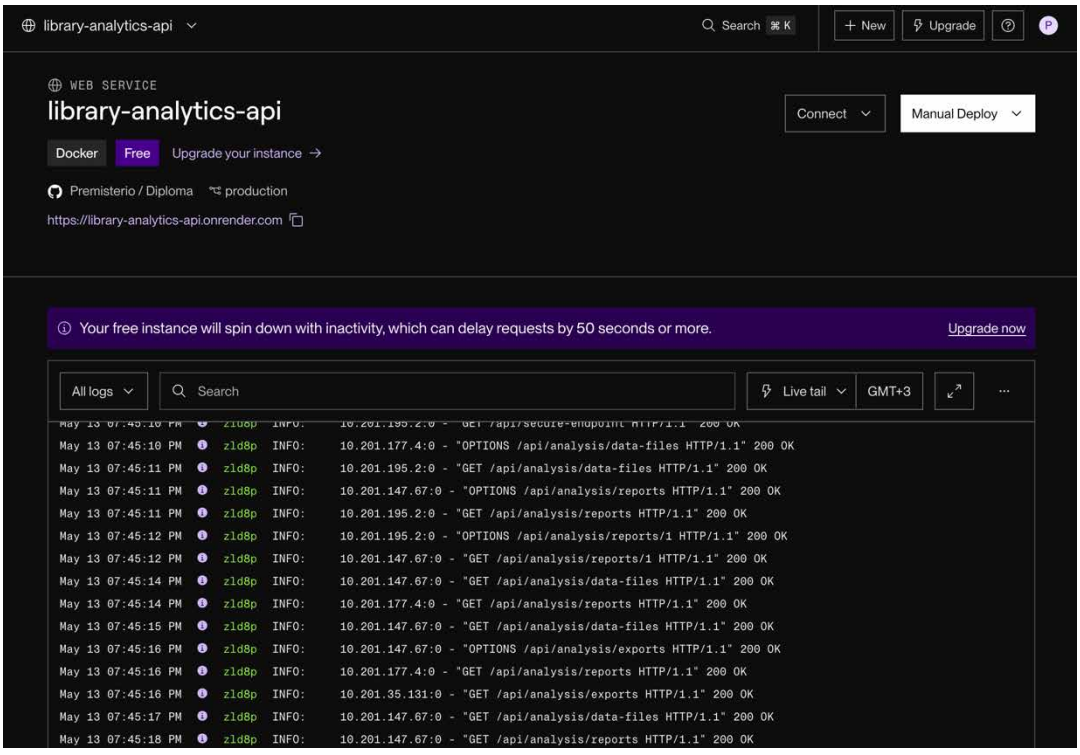


Рисунок 3.14 – Розгортання серверної частини системи на Render.com

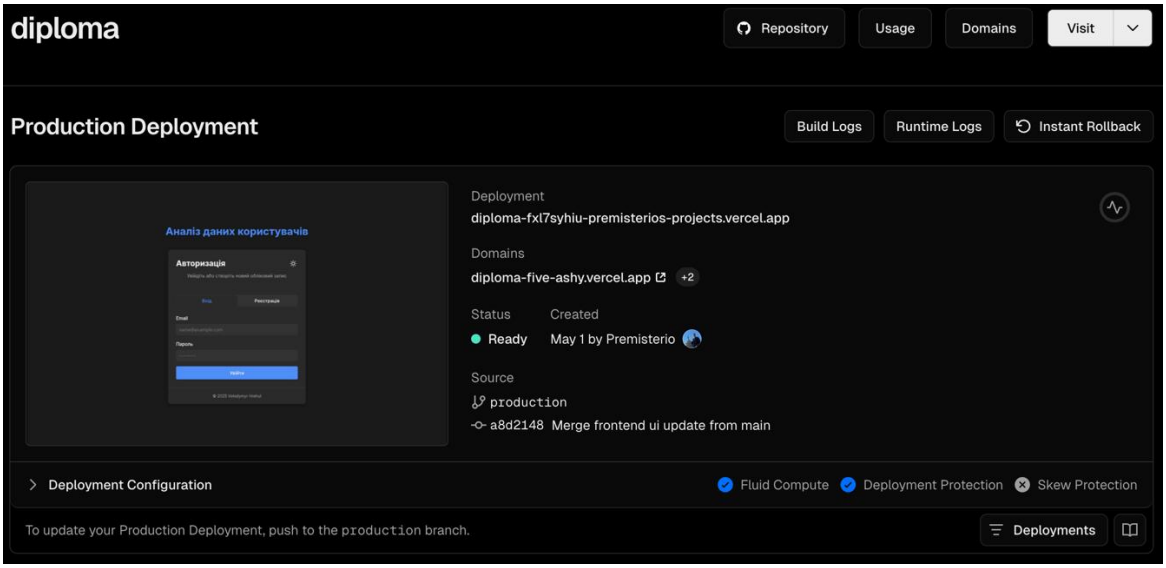


Рисунок 3.15 – Розгортання інтерфейсної частини системи на Vercel.com

РОЗДІЛ 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

Метою цього розділу є надання практичних рекомендацій щодо налаштування, встановлення, безпечного розгортання та подальшої експлуатації розробленої системи. Наведені поради будуть корисні як технічним спеціалістам, так і керівникам відділів IT.

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної роботи системи аналітики рекомендовано дотримуватись нижче вказаних технічних вимог до серверного та клієнтського обладнання.

Серверна частина (розгортання FastAPI + PostgreSQL):

- Процесор (CPU): не менше 2 ядер (рекомендовано 4 ядра і вище).
- Оперативна пам'ять (RAM): мінімум 4 ГБ (рекомендовано 8–16 ГБ).
- Дисковий простір: від 10 ГБ (SSD бажано, залежить від обсягів завантажуваних даних).
- Мережеве з'єднання: стабільне, з підтримкою HTTPS.
- Операційна система: Ubuntu 20.04+ / Debian / Windows Server 2019+.

Клієнтська частина (робота з React-інтерфейсом):

- Підтримувані браузери: Google Chrome (рекомендовано), Mozilla Firefox, Safari, Edge.
- Роздільна здатність екрана: мінімум 1280×720 (адаптивний інтерфейс підтримує будь-які сучасні екрани).
- Операційна система користувача: Windows 10+, macOS 11+, Linux (будь-який дистрибутив).

Програмні версії компонентів зазначені у розділі 3, таблиці 3.1.

4.2 Рекомендації щодо безпечного розгортання

Під час впровадження розробленого рішення основну увагу слід приділити захисту взаємодії між компонентами, контролю доступу та конфіденційності даних. Наведено технічні заходи, які реалізовано для безпечного розгортання як у локальному середовищі, так і в хмарі.

4.2.1 Захист транспортного рівня

Взаємодія між клієнтом і сервером слід здійснювати виключно через HTTPS. У хмарних середовищах (Render, Vercel) слід використовувати автоматично згенеровані SSL-сертифікати, щоб знизити ризик помилок при налаштуванні.

4.2.2 Діаграма розгортання

Діаграма розгортання відображає фізичне розміщення компонентів системи у мережевому середовищі, описуючи, які програмні модулі розміщуються на яких пристроях, та як між собою взаємодіють основні елементи інфраструктури.

У розробленій системі аналітики для онлайн-бібліотеки передбачено типову клієнт-серверну архітектуру, де клієнт взаємодіє із сервером через веб-браузер, а серверна частина розміщена у контейнері Docker на хмарній платформі. На Рисунку 4.1 представлено детальну структуру розгортання програмного забезпечення:

- Користувач (аналітик) взаємодіє із системою за допомогою персонального комп’ютера, оснащеного стандартним веб-браузером.
- Веб-браузер надсилає HTTPS-запити до веб-сервера, на якому розміщено середовище виконання FastAPI.

Всередині веб-сервера працює Docker-контейнер, який ізольовано містить усі необхідні компоненти:

- серверне застосування (App),
- базу даних PostgreSQL.

Комунікація між клієнтським і серверним середовищем відбувається за захищеним протоколом HTTPS, що гарантує шифрування даних та безпечну передачу авторизаційних токенів (JWT).

База даних PostgreSQL зберігає інформацію про користувачів, завантажені файли, звіти та експорти, забезпечуючи цілісність і доступність даних.

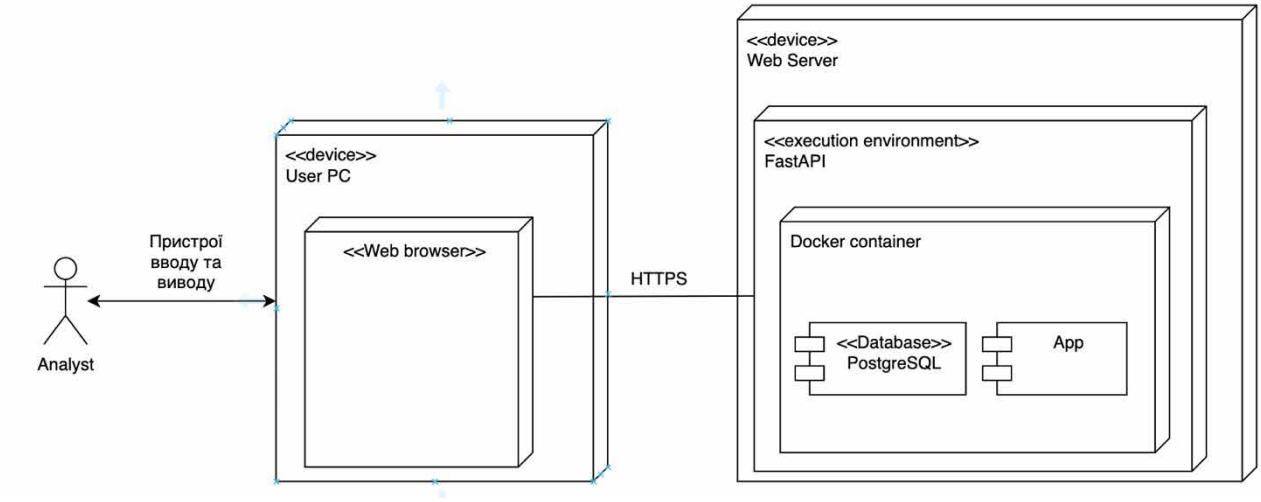


Рисунок 4.1 – Діаграма розгортання

4.2.3 Управління конфіденційними даними

Також обов’язково забезпечити розділення коду та налаштувань:

Дані облікових записів зберігаються у змінних середовища.

Ключові конфігурації винесено з коду, тобто чутливі файли мають бути додані в .gitignore, щоб вони не потрапили у публічний репозиторій.

У хмарі використано шифровані сховища для змінних середовища (див. рисунок 4.2).

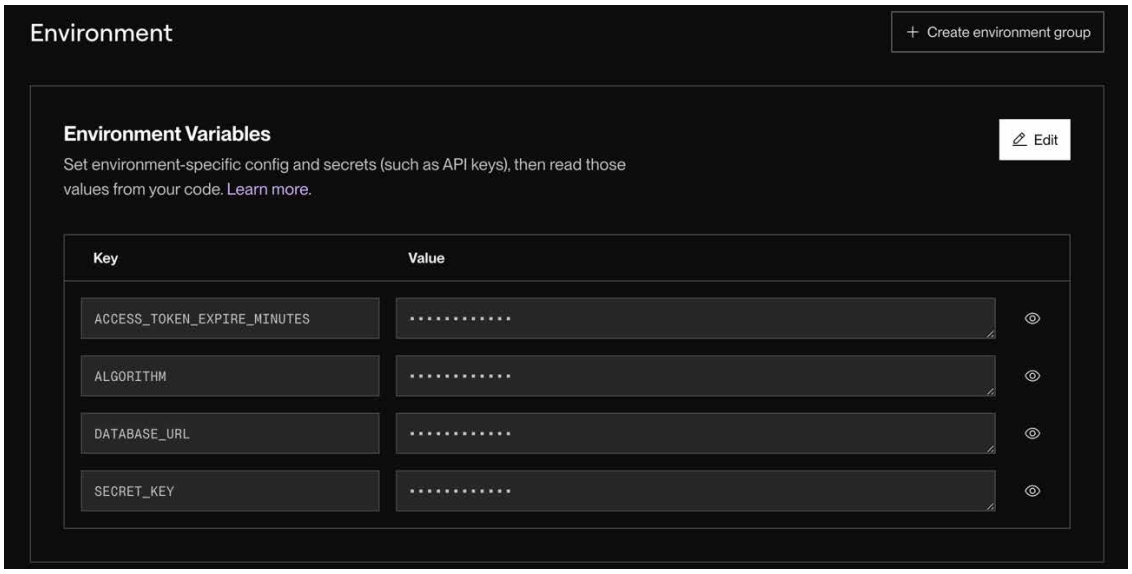


Рисунок 4.2 – Налаштування секретних ключів на Render

4.3 Технічне обслуговування

Для підтримки актуального стану безпеки необхідно постійно оновлювати програмне забезпечення та проводити аудит залежностей. У інтерфейсній частині ПЗ це можна зробити за допомогою команди: *npm run audit*. Ця команда автоматично проведе аудит всіх залежностей інтерфейсної частини системи. Також слід упровадити наступні заходи безпеки:

- Автоматизоване оновлення компонентів з мінімальними змінами версій
- Регламентоване резервне копіювання бази даних через заплановані задачі з часовими мітками

Після розгортання системи впроваджено комплексний підхід до моніторингу через інтеграцію з панеллю керування Render для відстеження стану основних компонентів, налаштування зовнішнього спостереження за доступністю сервісів, а також реєстрацію всіх спроб несанкціонованого доступу з метою оперативного реагування на потенційні загрози безпеці, що в сукупності

забезпечує надійний захист функціональності системи навіть у середовищі з відкритим мережевим доступом.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було успішно реалізовано програмне забезпечення для аналізу даних користувачів онлайн-бібліотеки, що дозволяє автоматизувати процес збору, обробки, візуалізації та експорту статистичної інформації. Система охоплює повний цикл роботи аналітика — від завантаження файлів до формування детальних звітів за ключовими метриками, такими як активність користувачів, популярність жанрів, динаміка відвідуваності тощо.

Програмне забезпечення реалізовано на основі сучасного технологічного стеку: серверна частина побудована з використанням Python і FastAPI, база даних — PostgreSQL, клієнтський інтерфейс — на основі React. Застосування контейнеризації через Docker забезпечило просте розгортання системи на локальному або хмарному середовищі. Також реалізовано інтерактивну API-документацію, зручну для тестування та розширення функціоналу.

Особливу увагу приділено безпеці: використано авторизацію через JWT-токени, захищено маршрути доступу, реалізовано валідацію вхідних даних, впроваджено механізми обмеження доступу до файлів і обробки помилок. Усі дії аналітика логуються, що дзволяє здійснювати аудит активності.

Розроблене ПЗ повністю відповідає функціональним та нефункціональним вимогам, задекларованим у першому розділі. Візуалізація результатів аналізу реалізована у вигляді інтерактивних графіків і таблиць, що дозволяє швидко і зручно інтерпретувати інформацію. Експорт доступний у форматах PDF, CSV, JSON, XLSX, що робить систему сумісною з іншими аналітичними інструментами.

Порівняльний аналіз із існуючими рішеннями (Koha, Alma, OpenBiblio, Tableau [6]) засвідчив, що створена система має оптимальне поєднання простоти

використання, гнучкості налаштування, продуктивності та доступності для невеликих організацій або бібліотек. Апробація системи у межах студентської конференції засвідчила її практичну цінність та актуальність у сфері цифрової трансформації освітньо-культурних установ. Також у Додатку Г зображено повну функціональність реалізованого графічного інтерфейсу.

Таким чином, результати дипломної роботи підтверджують доцільність та ефективність створеної системи, яка здатна підвищити рівень інформаційної підтримки управлінських рішень в онлайн-бібліотеках, адаптуючись до вимог часу та запитів користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрій Денисенко. Вступ до REST API – RESTful вебсервіси. *robot_dreams - онлайн-курси для фахівців у сфері big data, machine learning, data science | Робот Дрімс*. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 14.05.2025).
2. Що таке UI дизайн та як створювати привабливий дизайн інтерфейсів. *ITSTEP Академія Online освіта*. URL: https://cloud.itstep.org/blog/introduction-to-ui-design-creating-effective-and-attractive-interfaces (дата звернення: 14.05.2025).
3. Юлія К. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. *https://dou.ua*. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 15.05.2025).
4. Ялівець І. Що таке PostgreSQL? Вступ, переваги та недоліки. *Guru99*. URL: https://www.guru99.com/uk/introduction-postgresql.html (дата звернення: 14.05.2025).
5. Beck K. Test driven development: by example (the addison-wesley signature series). Addison-Wesley Professional, 2002. 240 p.
6. Business intelligence and analytics software | tableau. *Tableau*. URL: https://www.tableau.com (дата звернення: 14.05.2025).
7. Cloud application platform | render. *Render*. URL: https://render.com (date of access: 14.05.2025).
8. Docker: accelerated container application development. *Docker*. URL: https://www.docker.com (дата звернення: 14.05.2025).
9. FastAPI documentation. *FastAPI*. URL: https://fastapi.tiangolo.com (дата звернення: 14.05.2025).

10. Fetch чи Axios: що обрати? - SeoSite. *SeoSite*. URL: https://alexsmokinof.lviv.ua/fetch-chy-axios-shho-obraty (дата звернення: 14.05.2025).
11. JSON web token introduction - jwt.io. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction (дата звернення: 14.05.2025).
12. Library management system : alma - ex libris. *Ex Libris*. URL: https://exlibrisgroup.com/products/alma-library-services-platform/ (дата звернення:: 14.05.2025).
13. Matplotlib documentation – visualization with python. *Matplotlib – Visualization with Python*. URL: https://matplotlib.org (дата звернення: 14.05.2025).
14. Official website of koha library software. *Official Website of Koha Library Software*. URL: https://koha-community.org (дата звернення: 14.05.2025).
15. OpenAPI specification - version 3.1.0 | swagger. *API Documentation & Design Tools for Teams | Swagger*. URL: https://swagger.io/specification/ (дата звернення: 14.05.2025).
16. OpenBiblio documentation. *OpenBiblio*. URL: https://openbiblio.sourceforge.net/indexen.html (дата звернення: 14.05.2025).
17. Postman: the world's leading API platform | sign up for free. *Postman API Platform*. URL: https://www.postman.com (дата звернення: 15.05.2025).
18. Projector Creative & Tech Institute. React: Що таке React? Як почати вивчати Реакт? Основні навички. *CASES*. URL: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення: 15.05.2025).

19. Quick start – react. *React*. URL: https://react.dev/learn (дата звернення: 14.05.2025).
20. RESTful API design. CreateSpace Independent Publishing Platform, 2016. 294 p.
21. Seaborn: statistical data visualization – seaborn 0.13.2 documentation. *seaborn: statistical data visualization – seaborn 0.13.2 documentation*. URL: https://seaborn.pydata.org (дата звернення: 14.05.2025).
22. SQLAlchemy documentation – sqlalchemy 2.0 documentation. *SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation*. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 14.05.2025).
23. Uvicorn documentation. *Uvicorn*. URL: https://www.uvicorn.org (дата звернення: 15.05.2025).
24. Vercel: Build and deploy the best web experiences with the Frontend Cloud – Vercel. *Vercel*. URL: https://vercel.com/ (дата звернення: 15.05.2025).
25. Web content accessibility guidelines (WCAG) 2.1. *W3C*. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 15.05.2025).
26. Welcome to pydantic – pydantic documentation. *Welcome to Pydantic - Pydantic*. URL: https://docs.pydantic.dev/latest/ (дата звернення: 14.05.2025).
27. Welcome to GitHub - github. URL: https://github.com (дата звернення 22.05.2025)
28. Welcome to Visual Studio Code - Code Editing. Redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com (дата звернення: 21.05.2025).
29. Welcome to JetBrains. PyCharm: The only Python IDE you need. *JetBrains*. URL: https://www.jetbrains.com/pycharm/ (дата зверненя: 21.05.2025).

30. Welcome to Node.js – Run JavaScript Everywhere. *Node.js documentation*. URL: https://nodejs.org/en (дата звернення: 21.05.2025).
31. Welcome to Vite! *vitejs documentation*. URL: https://vite.dev (date of access: 21.05.2025).

ДОДАТОК А

SQL для створення таблиць БД:

```sql
-- Create tables based on your SQLAlchemy models
CREATE TABLE IF NOT EXISTS analysts (
    id SERIAL PRIMARY KEY,
    analyst_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    hashed_password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS data_files (
    id SERIAL PRIMARY KEY,
    analyst_id INTEGER REFERENCES analysts(id),
    filename VARCHAR(255) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    upload_date TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS analysis_reports (
    id SERIAL PRIMARY KEY,
    analyst_id INTEGER REFERENCES analysts(id),
    report_name VARCHAR(255) NOT NULL,
    report_data JSONB NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS report_exports (
    id SERIAL PRIMARY KEY,
    report_id INTEGER REFERENCES analysis_reports(id),
    analyst_id INTEGER REFERENCES analysts(id),
    export_format VARCHAR(10) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

ДОДАТОК Б

Відрізок коду логіки авторизації користувача:

auth.py:

```python
from datetime import datetime, timedelta, timezone
from dotenv import load_dotenv
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt
from sqlalchemy.orm import Session
from app.core.database import get_db
from app.models.analyst import Analyst
import os

load_dotenv()

SECRET_KEY = os.getenv("SECRET_KEY")
ALGORITHM = os.getenv("ALGORITHM", "HS256")
ACCESS_TOKEN_EXPIRE_MINUTES = int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "1440"))

def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (expires_delta or timedelta(minutes=15))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def create_refresh_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (expires_delta or timedelta(days=7))
    to_encode.update({"exp": expire, "type": "refresh"})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def decode_token(token: str):
    try:
        return jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
    except JWTError:
        return None
```

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/login")

def get_current_analyst(
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db)
) -> Analyst:
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        analyst_id: str = payload.get("sub")
        if analyst_id is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    analyst = db.query(Analyst).filter(Analyst.id == int(analyst_id)).first()
    if analyst is None:
        raise credentials_exception
    return analyst
```

Відрізок логіки створення моделі користувача:

analyst.py:

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import relationship
from app.core.database import Base

class Analyst(Base):
    __tablename__ = "analysts"

    id = Column(Integer, primary_key=True, index=True)
    analyst_name = Column(String, index=True, nullable=False)
```

```
    email = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)

    reports = relationship("AnalysisReport", back_populates="analyst")
    data_files = relationship("DataFile", back_populates="analyst")
    exports = relationship("ReportExport", back_populates="analyst")
```

Алгоритм завантаження файлів користувача:

file_upload.py:

```
import os
import uuid
import shutil
from fastapi import UploadFile, HTTPException
from typing import Optional

# Define upload directory relative to project root
UPLOAD_DIR = os.path.join("uploads", "data_files")

# Ensure upload directory exists
os.makedirs(UPLOAD_DIR, exist_ok=True)

def save_upload_file(upload_file: UploadFile, analyst_id: int) -> str:
    """
    Save an uploaded file to the file system
    Returns the file path relative to the project root
    """
    # Create a unique filename to avoid collisions
    file_extension = os.path.splitext(upload_file.filename)[1]
    unique_filename = f"{uuid.uuid4()}{file_extension}"

    # Create directories if they don't exist
    analyst_upload_dir = os.path.join(UPLOAD_DIR, str(analyst_id))
    os.makedirs(analyst_upload_dir, exist_ok=True)

    # Create full file path
    file_path = os.path.join(analyst_upload_dir, unique_filename)
```

```
    try:
        # Save the file
        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(upload_file.file, buffer)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Could not save file: {str(e)}")
    finally:
        upload_file.file.close()

    return file_path

def get_file_path(filename: str, analyst_id: int) -> Optional[str]:
    """
    Get the full file path for a file
    Returns None if file doesn't exist
    """
    file_path = os.path.join(UPLOAD_DIR, str(analyst_id), filename)
    if os.path.exists(file_path):
        return file_path
    return None
```

Відрізок коду з алгоритмом фільтрування даних та генерації аналізу: library_analysis.py:

```
import json
import os
from sqlalchemy.orm import Session
from typing import Optional, Dict, Any, List
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from collections import Counter, defaultdict
import re

from app.models.library_data import AnalysisReport, DataFile
from app.models.analyst import Analyst

class LibraryDataAnalyzer:
```

```
def __init__(self, data_path):
    """Initialize with path to data.json file"""
    self.load_data(data_path)
    self.initialize_dataframes()

def load_data(self, data_path):
    """Load JSON data from file"""
    with open(data_path, 'r') as file:
        self.raw_data = json.load(file)
    self.users = self.raw_data['users']

def initialize_dataframes(self):
    """Convert nested JSON to pandas DataFrames for easier analysis"""
    # Core user data
    user_base_data = []
    for user in self.users:
        base_data = {
            'user_id': user['user_id'],
            'registration_date': user['account']['registration_date'],
            'account_type': user['account']['account_type'],
            'subscription_status': user['account']['subscription_status'],
            'login_frequency': user['account']['login_frequency'],
            'last_login': user['account']['last_login'],
            'age_range': user['profile']['age_range'],
            'education_level': user['profile']['education_level'],
            'profession': user['profile']['profession']
        }
        user_base_data.append(base_data)
    self.user_df = pd.DataFrame(user_base_data)

    # Book borrowing data
    borrowing_data = []
    for user in self.users:
        user_id = user['user_id']
        for book in user['activity'].get('books_borrowed', []):
            book_data = {
                'user_id': user_id,
                'book_id': book['book_id'],
                'title': book['title'],
```

```
            'author': book['author'],
            'genre': book['genre'],
            'borrowed_date': book['borrowed_date'],
            'return_date': book.get('return_date'),
            'rating': book.get('rating'),
            'completed': book.get('completed', False)
        }
        borrowing_data.append(book_data)
self.borrowing_df = pd.DataFrame(borrowing_data)

# Reading sessions data
session_data = []
for user in self.users:
    user_id = user['user_id']
    for session in user['activity'].get('reading_sessions', []):
        session_info = {
            'user_id': user_id,
            'book_id': session['book_id'],
            'date': session['date'],
            'duration_minutes': session['duration_minutes'],
            'pages_read': session['pages_read'],
            'device': session['device']
        }
        session_data.append(session_info)
self.session_df = pd.DataFrame(session_data)

# Search history data
search_data = []
for user in self.users:
    user_id = user['user_id']
    for search in user['activity'].get('search_history', []):
        search_info = {
            'user_id': user_id,
            'timestamp': search['timestamp'],
            'query': search['query']
        }
        search_data.append(search_info)
self.search_df = pd.DataFrame(search_data)
```

```
    # Process datetime columns in all dataframes
    self.process_datetime_columns()

def process_datetime_columns(self):
    """Convert string datetime to datetime objects"""
    # Process user dataframe
    datetime_cols = ['registration_date', 'last_login']
    for col in datetime_cols:
        if col in self.user_df.columns:
            # Parse with timezone information, then convert to naive datetime
            self.user_df[col] = pd.to_datetime(self.user_df[col]).dt.tz_localize(None)

    # Process borrowing dataframe
    if not self.borrowing_df.empty:
        for col in ['borrowed_date', 'return_date']:
            if col in self.borrowing_df.columns:
                self.borrowing_df[col] = pd.to_datetime(self.borrowing_df[col]).dt.tz_localize(None)

    # Process session dataframe
    if not self.session_df.empty:
        if 'date' in self.session_df.columns:
            self.session_df['date'] = pd.to_datetime(self.session_df['date']).dt.tz_localize(None)

    # Process search dataframe
    if not self.search_df.empty:
        if 'timestamp' in self.search_df.columns:
            self.search_df['timestamp'] = pd.to_datetime(self.search_df['timestamp']).dt.tz_localize(None)

# Analysis methods (same as in your original app.py)
def analyze_usage_patterns(self):
    """Analyze user activity patterns"""
    results = {}

    if not self.session_df.empty:
        # Daily activity pattern (hour of day)
        self.session_df['hour'] = self.session_df['date'].dt.hour
        hourly_activity = self.session_df['hour'].value_counts().sort_index()
        results['hourly_activity'] = hourly_activity.to_dict()
```

```
        # Weekly pattern
        self.session_df['day_of_week'] = self.session_df['date'].dt.day_name()
        day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        weekly_activity = self.session_df['day_of_week'].value_counts()
        weekly_activity = weekly_activity.reindex(day_order)
        results['weekly_activity'] = weekly_activity.to_dict()

        # Average session duration by device
        avg_duration = self.session_df.groupby('device')['duration_minutes'].mean()
        results['avg_duration_by_device'] = avg_duration.to_dict()

        # Average pages read per session
        avg_pages = self.session_df.groupby('device')['pages_read'].mean()
        results['avg_pages_by_device'] = avg_pages.to_dict()

    # User activity recency
    self.user_df['last_login_naive'] = self.user_df['last_login'].dt.tz_localize(None)
    self.user_df['days_since_login'] = (datetime.now() - self.user_df['last_login_naive']).dt.days

    recency_segments = pd.cut(self.user_df['days_since_login'],
                              bins=[0, 7, 30, 90, float('inf')],
                              labels=['Last 7 days', '8-30 days', '31-90 days', '90+ days'])
    recency_counts = recency_segments.value_counts()
    results['login_recency'] = recency_counts.to_dict()

    return results

def analyze_content_performance(self):
    """Analyze book performance metrics"""
    results = {}

    if not self.borrowing_df.empty:
        # Most borrowed books
        top_books = self.borrowing_df['title'].value_counts().head(10)
        results['top_borrowed_books'] = top_books.to_dict()

        # Genre popularity
        genre_popularity = self.borrowing_df['genre'].value_counts()
        results['genre_popularity'] = genre_popularity.to_dict()
```

```
        # Average ratings by genre
        avg_ratings = self.borrowing_df.groupby('genre')['rating'].mean().round(2)
        results['avg_ratings_by_genre'] = avg_ratings.to_dict()

        # Completion rates by genre
        completion_rates = self.borrowing_df.groupby('genre')['completed'].mean().round(2)
        results['completion_rates'] = completion_rates.to_dict()

        # Top authors
        top_authors = self.borrowing_df['author'].value_counts().head(10)
        results['top_authors'] = top_authors.to_dict()

    return results

def analyze_user_segments(self):
    """Segment users based on behavior and demographics"""
    results = {}

    # Segment by account type
    account_distribution = self.user_df['account_type'].value_counts()
    results['account_type_distribution'] = account_distribution.to_dict()

    # Segment by age range
    age_distribution = self.user_df['age_range'].value_counts()
    results['age_distribution'] = age_distribution.to_dict()

    # Segment by education
    education_distribution = self.user_df['education_level'].value_counts()
    results['education_distribution'] = education_distribution.to_dict()

    # Segment by profession
    profession_distribution = self.user_df['profession'].value_counts().head(10)
    results['top_professions'] = profession_distribution.to_dict()

    # Cross-analyze age and content preferences
    if not self.borrowing_df.empty:
        # Merge user and borrowing data
        user_borrow = pd.merge(self.borrowing_df, self.user_df[['user_id', 'age_range']], on='user_id')
```

```
        genre_by_age = user_borrow.groupby(['age_range', 'genre']).size().unstack().fillna(0)

        # Convert to percentages within each age group
        genre_by_age_pct = genre_by_age.div(genre_by_age.sum(axis=1), axis=0).round(2)
        results['genre_preferences_by_age'] = genre_by_age_pct.to_dict()

    return results

def analyze_search_patterns(self):
    """Analyze user search behavior"""
    results = {}

    if not self.search_df.empty:
        # Extract common keywords from searches
        all_queries = ' '.join(self.search_df['query'].tolist()).lower()
        # Remove stopwords (simplified approach)
        stopwords = ['the', 'a', 'an', 'and', 'in', 'on', 'at', 'for', 'to', 'of', 'with', 'by']
        query_words = [word for word in re.findall(r'\b\w+\b', all_queries) if word not in stopwords]

        word_freq = Counter(query_words).most_common(20)
        results['top_search_terms'] = dict(word_freq)

        # Search volume by hour of day
        self.search_df['hour'] = self.search_df['timestamp'].dt.hour
        search_by_hour = self.search_df['hour'].value_counts().sort_index()
        results['searches_by_hour'] = search_by_hour.to_dict()

    return results

def analyze_retention(self):
    """Analyze user retention metrics"""
    results = {}

    # Calculate account age
    self.user_df['registration_date_naive'] = self.user_df['registration_date'].dt.tz_localize(None)
    self.user_df['account_age_days'] = (datetime.now() - self.user_df['registration_date_naive']).dt.days

    # Segment by account age
    age_bins = [0, 30, 90, 180, 365, float('inf')]
```

```
        age_labels = ['< 1 month', '1-3 months', '3-6 months', '6-12 months', '> 1 year']
        self.user_df['account_age_segment'] = pd.cut(self.user_df['account_age_days'],
                                                     bins=age_bins,
                                                     labels=age_labels)

        tenure_dist = self.user_df['account_age_segment'].value_counts()
        results['user_tenure_distribution'] = tenure_dist.to_dict()

        # Activity by tenure
        if not self.session_df.empty:
            user_activity = self.session_df.groupby('user_id').size().reset_index(name='activity_count')
            user_tenure = pd.merge(user_activity,
                                   self.user_df[['user_id', 'account_age_segment']],
                                   on='user_id')
            activity_by_tenure = user_tenure.groupby('account_age_segment', observed=False)[
                'activity_count'].mean().round(1)
            results['avg_activity_by_tenure'] = activity_by_tenure.to_dict()

        return results

    def generate_comprehensive_report(self):
        """Generate a comprehensive analysis report"""
        report = {
            'report_date': datetime.now().strftime('%Y-%m-%d'),
            'total_users': len(self.user_df),
            'usage_patterns': self.analyze_usage_patterns(),
            'content_performance': self.analyze_content_performance(),
            'user_segments': self.analyze_user_segments(),
            'search_patterns': self.analyze_search_patterns(),
            'retention_metrics': self.analyze_retention()
        }

        return report

def create_data_file(db: Session, file_path: str, analyst_id: int, filename: str):
    """Store information about an uploaded data file"""
    data_file = DataFile(
        analyst_id=analyst_id,
        filename=filename,
```

```
        file_path=file_path
    )
    db.add(data_file)
    db.commit()
    db.refresh(data_file)
    return data_file

def get_data_files_by_analyst(db: Session, analyst_id: int):
    """Get all data files uploaded by an analyst"""
    return db.query(DataFile).filter(DataFile.analyst_id == analyst_id).all()

def save_analysis_report(db: Session, report_name: str, report_data: Dict, analyst_id: int):
    """Save analysis report to database"""
    report = AnalysisReport(
        analyst_id=analyst_id,
        report_name=report_name,
        report_data=report_data
    )
    db.add(report)
    db.commit()
    db.refresh(report)
    return report

def get_reports_by_analyst(db: Session, analyst_id: int):
    """Get all reports created by an analyst"""
    return db.query(AnalysisReport).filter(AnalysisReport.analyst_id == analyst_id).all()

def get_report_by_id(db: Session, report_id: int):
    """Get a specific report by ID"""
    return db.query(AnalysisReport).filter(AnalysisReport.id == report_id).first()
```

ДОДАТОК В

Алгоритм формування статистичного звіту для експорту:

export_report.py:

```
import os
import pandas as pd
import json
from reportlab.lib.pagesizes import letter, landscape
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib import colors

def export_to_pdf(report_data, file_path):
    """
    Export report data to a PDF file using ReportLab

    Args:
        report_data (dict): Report data dictionary
        file_path (str): Path where PDF will be saved
    """
    doc = SimpleDocTemplate(file_path, pagesize=landscape(letter))  # Changed to landscape for wider tables
    styles = getSampleStyleSheet()
    elements = []

    # Add title
    title = Paragraph(f"Library Analysis Report: {report_data.get('report_name', 'Unnamed')}", styles['Title'])
    elements.append(title)
    elements.append(Spacer(1, 12))

    # Add report date and total users
    elements.append(Paragraph(f"Report Date: {report_data.get('report_date', 'N/A')}", styles['Normal']))
    elements.append(Paragraph(f"Total Users: {report_data.get('total_users', 0)}", styles['Normal']))
    elements.append(Spacer(1, 12))

    # Add usage patterns section
    if 'usage_patterns' in report_data:
        elements.append(Paragraph("Usage Patterns", styles['Heading2']))
        elements.append(Spacer(1, 6))
```

```
    for key, value in report_data['usage_patterns'].items():
        elements.append(Paragraph(key, styles['Heading3']))

        if isinstance(value, dict):
            data = [["Item", "Value"]]
            for sub_key, sub_value in value.items():
                # Format complex values
                if isinstance(sub_value, dict):
                    formatted_value = format_dict_for_display(sub_value)
                else:
                    formatted_value = str(sub_value)

                data.append([sub_key, formatted_value])

            table = Table(data, colWidths=[200, 400])  # Wider value column for complex data
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (1, 0), colors.grey),
                ('TEXTCOLOR', (0, 0), (1, 0), colors.whitesmoke),
                ('ALIGN', (0, 0), (0, -1), 'LEFT'),  # Left align items
                ('ALIGN', (1, 0), (1, -1), 'LEFT'),  # Left align values for readability
                ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
                ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))

    # Add content performance section
    if 'content_performance' in report_data:
        elements.append(Paragraph("Content Performance", styles['Heading2']))
        elements.append(Spacer(1, 6))

        for key, value in report_data['content_performance'].items():
            elements.append(Paragraph(key, styles['Heading3']))

            if isinstance(value, dict):
                data = [["Item", "Value"]]
                for sub_key, sub_value in value.items():
```

```
            # Format complex values
            if isinstance(sub_value, dict):
                formatted_value = format_dict_for_display(sub_value)
            else:
                formatted_value = str(sub_value)

            data.append([sub_key, formatted_value])

        table = Table(data, colWidths=[200, 400])
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (0, -1), 'LEFT'),
            ('ALIGN', (1, 0), (1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('GRID', (0, 0), (-1, -1), 1, colors.black)
        ]))
        elements.append(table)
        elements.append(Spacer(1, 12))

# Add user segments section
if 'user_segments' in report_data:
    elements.append(Paragraph("User Segments", styles['Heading2']))
    elements.append(Spacer(1, 6))

    for key, value in report_data['user_segments'].items():
        elements.append(Paragraph(key, styles['Heading3']))

        # Special handling for genre_preferences_by_age
        if key == 'genre_preferences_by_age':
            # Create a more structured table for age preferences
            if isinstance(value, dict):
                # Get all age groups
                age_groups = set()
                genres = list(value.keys())

                # Find all age groups across all genres
                for genre_data in value.values():
```

```
            if isinstance(genre_data, dict):
                for age_group in genre_data.keys():
                    age_groups.add(age_group)

        age_groups = sorted(list(age_groups))

        # Create header row with age groups
        header_row = ["Genre"] + age_groups
        data = [header_row]

        # Add data for each genre
        for genre, age_data in value.items():
            if isinstance(age_data, dict):
                row = [genre]
                for age_group in age_groups:
                    preference = age_data.get(age_group, "N/A")
                    row.append(str(preference))
                data.append(row)

        # Calculate column widths based on number of columns
        col_count = len(header_row)
        available_width = 600  # Total available width
        genre_col_width = 120  # Fixed width for genre column
        age_col_width = (available_width - genre_col_width) / (col_count - 1)
        col_widths = [genre_col_width] + [age_col_width] * (col_count - 1)

        table = Table(data, colWidths=col_widths)
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('GRID', (0, 0), (-1, -1), 1, colors.black)
        ]))
        elements.append(table)
        elements.append(Spacer(1, 12))
    else:
        # Standard table processing for other user segments data
```

```
            if isinstance(value, dict):
                data = [["Item", "Value"]]
                for sub_key, sub_value in value.items():
                    # Format complex values
                    if isinstance(sub_value, dict):
                        formatted_value = format_dict_for_display(sub_value)
                    else:
                        formatted_value = str(sub_value)

                    data.append([sub_key, formatted_value])

                table = Table(data, colWidths=[200, 400])
                table.setStyle(TableStyle([
                    ('BACKGROUND', (0, 0), (1, 0), colors.grey),
                    ('TEXTCOLOR', (0, 0), (1, 0), colors.whitesmoke),
                    ('ALIGN', (0, 0), (0, -1), 'LEFT'),
                    ('ALIGN', (1, 0), (1, -1), 'LEFT'),
                    ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
                    ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
                    ('GRID', (0, 0), (-1, -1), 1, colors.black)
                ]))
                elements.append(table)
                elements.append(Spacer(1, 12))

    # Build the PDF
    doc.build(elements)

def format_dict_for_display(d):
    """Format a dictionary for better display in tables"""
    formatted_parts = []
    for k, v in d.items():
        formatted_parts.append(f"{k}: {v}")
    return ", ".join(formatted_parts)

def export_to_excel(report_data, file_path):
    """
    Export report data to an Excel file with multiple sheets

    Args:
```

```
    report_data (dict): Report data dictionary
    file_path (str): Path where Excel will be saved
    """
    with pd.ExcelWriter(file_path, engine='openpyxl') as writer:
        # Overview sheet
        overview_data = {
            'Metric': ['Report Name', 'Report Date', 'Total Users'],
            'Value': [
                report_data.get('report_name', 'Unnamed'),
                report_data.get('report_date', ''),
                report_data.get('total_users', 0)
            ]
        }
        pd.DataFrame(overview_data).to_excel(writer, sheet_name='Overview', index=False)

        # Usage Patterns sheet
        if 'usage_patterns' in report_data:
            usage_data = []
            for key, value in report_data['usage_patterns'].items():
                if isinstance(value, dict):
                    for sub_key, sub_value in value.items():
                        usage_data.append({
                            'Category': key,
                            'Item': sub_key,
                            'Value': sub_value if not isinstance(sub_value, dict) else json.dumps(sub_value)
                        })
            if usage_data:
                pd.DataFrame(usage_data).to_excel(writer, sheet_name='Usage Patterns', index=False)

        # Content Performance sheet
        if 'content_performance' in report_data:
            content_data = []
            for key, value in report_data['content_performance'].items():
                if isinstance(value, dict):
                    for sub_key, sub_value in value.items():
                        content_data.append({
                            'Category': key,
                            'Item': sub_key,
                            'Value': sub_value if not isinstance(sub_value, dict) else json.dumps(sub_value)
```

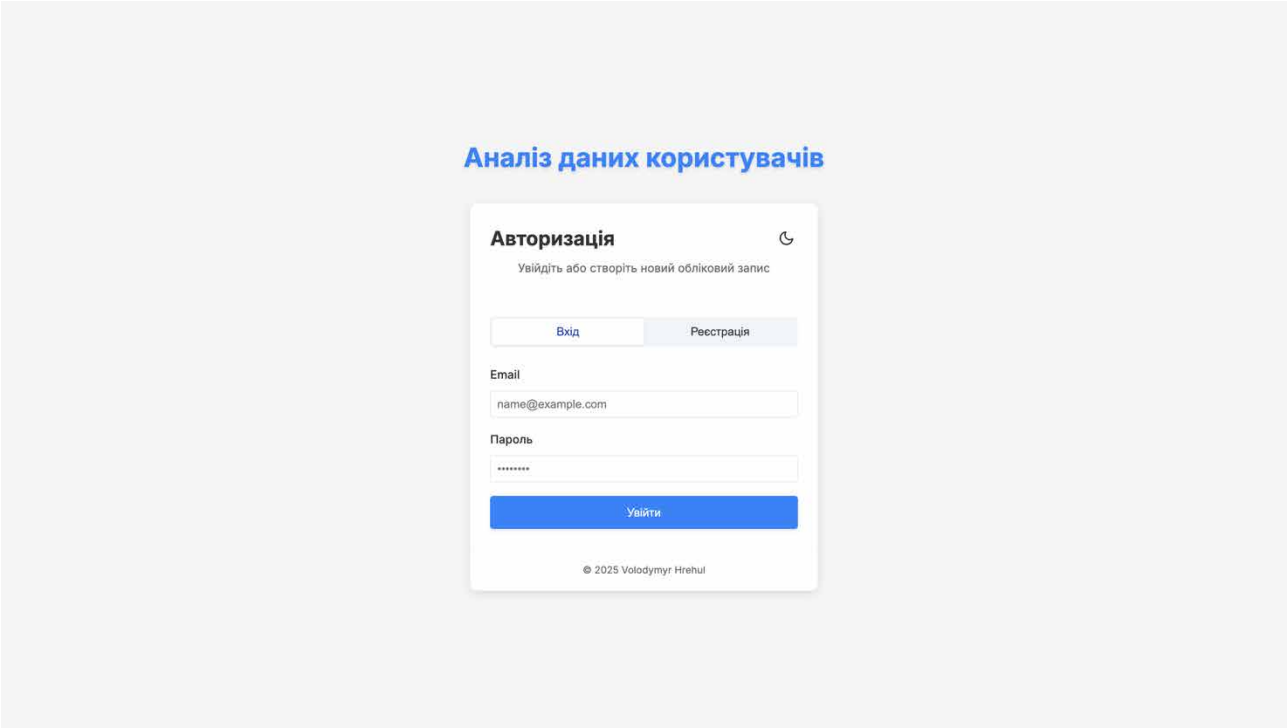
```
            })
    if content_data:
        pd.DataFrame(content_data).to_excel(writer, sheet_name='Content Performance', index=False)

# User Segments sheet
if 'user_segments' in report_data:
    # General user segments data
    segments_data = []

    # Handle regular data and separate genre_preferences_by_age
    for key, value in report_data['user_segments'].items():
        if key == 'genre_preferences_by_age' and isinstance(value, dict):
            # Create special sheet for genre preferences by age
            # First prepare the data as a proper DataFrame
            genres = []
            age_data = {}

            for genre, prefs in value.items():
                genres.append(genre)
                if isinstance(prefs, dict):
                    for age_group, score in prefs.items():
                        if age_group not in age_data:
                            age_data[age_group] = {}
                        age_data[age_group][genre] = score

            # Create a DataFrame for each age group
            if age_data:
                age_df = pd.DataFrame(age_data).T
                age_df.index.name = 'Age Group'
                age_df.to_excel(writer, sheet_name='Genre Prefs by Age')
        else:
            # Regular data handling
            if isinstance(value, dict):
                for sub_key, sub_value in value.items():
                    segments_data.append({
                        'Category': key,
                        'Item': sub_key,
                        'Value': sub_value if not isinstance(sub_value, dict) else json.dumps(sub_value)
                    })
```

```
    if segments_data:
        pd.DataFrame(segments_data).to_excel(writer, sheet_name='User Segments', index=False)
```

ДОДАТОК Г

Реалізований графічний інтерфейс системи за допомогою React:

Сторінка авторизації:



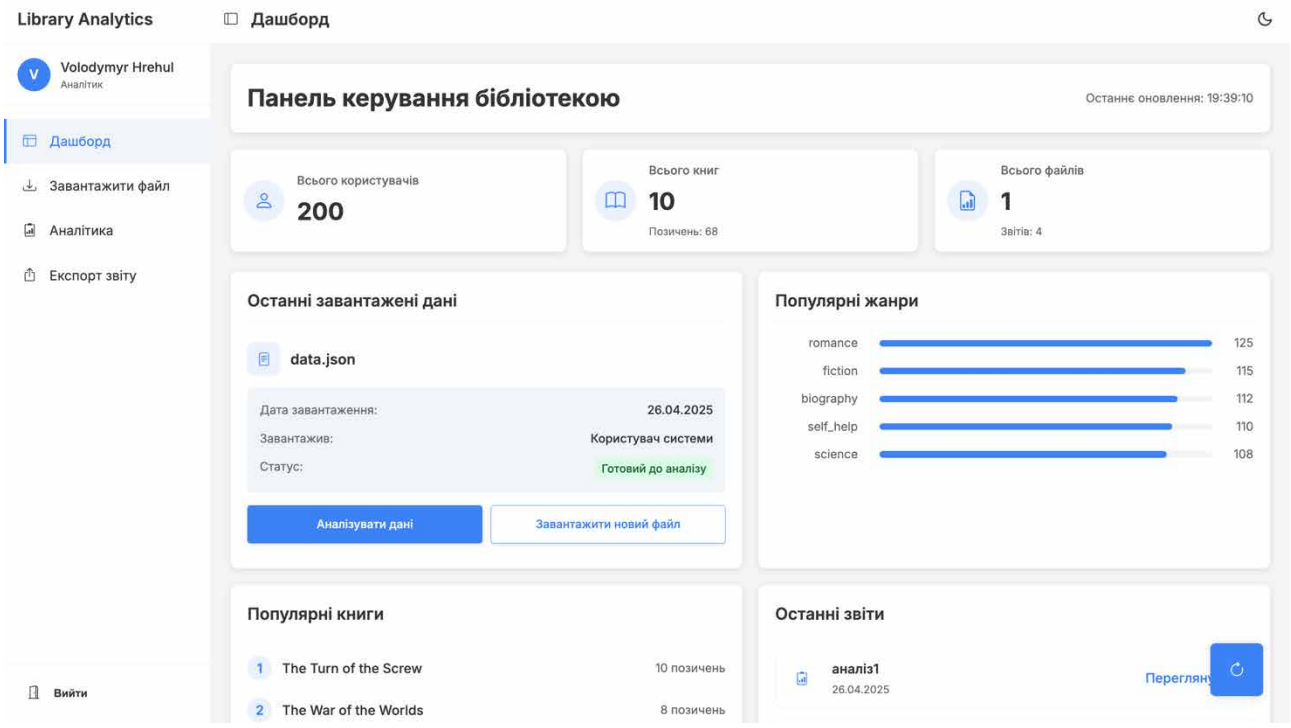
Код реалізованого інтерфейсу на React:

AuthPage.jsx:

```
import { useState } from "react"
import LoginForm from "../components/LoginForm"
import RegisterForm from "../components/RegisterForm"
import ThemeToggle from "../components/ThemeToggle"

function AuthPage() {
 const [activeTab, setActiveTab] = useState("login")

 return (
  <div className="auth-container">
   <h1 className="app-title">Аналіз даних користувачів</h1>
   <div className="auth-card">
    <div className="auth-header">
```

```
        <div className="header-top">
          <h2 className="auth-title">Авторизація</h2>
          <ThemeToggle />
        </div>
        <p className="auth-description">Увійдіть або створіть новий обліковий запис</p>
      </div>

      <div className="auth-content">
        <div className="tabs">
          <div className="tabs-list">
            <button
              className={`tab-trigger ${activeTab === "login" ? "active" : ""}`}
              onClick={() => setActiveTab("login")}
            >
              Вхід
            </button>
            <button
              className={`tab-trigger ${activeTab === "register" ? "active" : ""}`}
              onClick={() => setActiveTab("register")}
            >
              Реєстрація
            </button>
          </div>
          {activeTab === "login" ? <LoginForm /> : <RegisterForm />}
        </div>
      </div>

      <div className="auth-footer">
        <p>© 2025 Volodymyr Hrehul</p>
      </div>
    </div>
  </div>
 )
}

export default AuthPage
```

Головна сторінка («Дашборд»):



Програмна реалізація:

Dashboard.jsx:

```
import { useState, useEffect } from "react";
import { fileAPI, analysisAPI } from "../services/api";
import LoadingSpinner from "../components/LoadingSpinner";
import Alert from "../components/Alert";
```

```
import { Link } from "react-router-dom";

function DashboardHome() {
 const [stats, setStats] = useState({
  totalFiles: 0,
  totalUsers: 0,
  popularGenres: [],
  recentActivity: [],
  recentReports: [],
  latestFile: null,
  totalBooks: 0,
  totalBorrows: 0,
  topBooks: []
 });

 const [loading, setLoading] = useState(true);
 const [error, setError] = useState("");
 const [lastUpdated, setLastUpdated] = useState(null);

 useEffect(() => {
  // Fetch dashboard data when component mounts
  fetchDashboardData();
 }, []);

 const fetchDashboardData = async () => {
  setLoading(true);
  setError("");

  try {
   // Fetch files
   const filesResponse = await fileAPI.getUploadedFiles();

   // Fetch reports
   const reportsResponse = await analysisAPI.getReports();

   if (filesResponse.success && reportsResponse.success) {
    const files = filesResponse.data;
    const reports = reportsResponse.data || [];
```

```
// Get the latest report to extract data
let popularGenres = [];
let userCount = 0;
let totalBooks = 0;
let totalBorrows = 0;
let topBooks = [];

if (reports.length > 0) {
  // Fetch the latest report's data
  const latestReportResponse = await analysisAPI.getReportById(reports[0].id);

  if (latestReportResponse.success && latestReportResponse.data.report_data) {
    const reportData = latestReportResponse.data.report_data;

    // Extract popular genres
    if (reportData.content_performance && reportData.content_performance.genre_popularity) {
      popularGenres = Object.entries(reportData.content_performance.genre_popularity)
        .map(([name, count]) => ({ name, count }))
        .sort((a, b) => b.count - a.count)
        .slice(0, 5);
    }

    // Extract user data
    if (reportData.total_users) {
      userCount = reportData.total_users;
    }

    // Extract book data
    if (reportData.content_performance) {
      const topBorrowedBooks = reportData.content_performance.top_borrowed_books || {};
      totalBooks = Object.keys(topBorrowedBooks).length;
      totalBorrows = Object.values(topBorrowedBooks).reduce((a, b) => a + b, 0);

      // Get top books
      topBooks = Object.entries(topBorrowedBooks)
        .map(([title, count]) => ({ title, count }))
        .sort((a, b) => b.count - a.count)
        .slice(0, 5);
    }
```

```
    }
  }

  // Create recent activity list from files and reports
  const recentActivity = [
    // Add recent file uploads
    ...files.slice(0, 5).map(file => ({
      user: file.uploader_email || "",
      action: "Користувач завантажив файл",
      time: formatTimeAgo(new Date(file.upload_date)),
      detail: file.filename
    })),

    // Add recent reports
    ...reports.slice(0, 5).map(report => ({
      user: report.analyst_email || "",
      action: "Користувач створив звіт",
      time: formatTimeAgo(new Date(report.created_at)),
      detail: report.report_name
    }))
  ].sort((a, b) => {
    // Convert relative time strings to sort order
    return getTimeWeight(a.time) - getTimeWeight(b.time);
  }).slice(0, 8);

  setStats({
    totalFiles: files.length,
    totalUsers: userCount,
    popularGenres: popularGenres,
    recentActivity: recentActivity,
    recentReports: reports.slice(0, 5),
    latestFile: files.length > 0 ? files[0] : null,
    totalBooks: totalBooks,
    totalBorrows: totalBorrows,
    topBooks: topBooks
  });

  setLastUpdated(new Date());
} else {
```

```
      setError(filesResponse.error || reportsResponse.error || "Не вдалося завантажити дані");
    }
  } catch (err) {
    console.error("Error fetching dashboard data:", err);
    setError("Помилка при завантаженні даних дашборду");
  } finally {
    setLoading(false);
  }
};

// Helper function to format time ago
const formatTimeAgo = (date) => {
  const now = new Date();
  const diffMs = now - date;
  const diffSec = Math.round(diffMs / 1000);
  const diffMin = Math.round(diffSec / 60);
  const diffHour = Math.round(diffMin / 60);
  const diffDay = Math.round(diffHour / 24);

  if (diffDay > 0) {
    return `${diffDay} ${pluralize(diffDay, 'день', 'дні', 'днів')} тому`;
  } else if (diffHour > 0) {
    return `${diffHour} ${pluralize(diffHour, 'година', 'години', 'годин')} тому`;
  } else if (diffMin > 0) {
    return `${diffMin} ${pluralize(diffMin, 'хвилина', 'хвилини', 'хвилин')} тому`;
  }
  return 'Щойно';
};

// Helper function for Ukrainian pluralization
const pluralize = (count, one, few, many) => {
  if (count % 10 === 1 && count % 100 !== 11) {
    return one;
  } else if (
    [2, 3, 4].includes(count % 10) &&
    ![12, 13, 14].includes(count % 100)
  ) {
    return few;
  } else {
```

```
    return many;
  }
};

// Helper function to assign a weight to time strings for sorting
const getTimeWeight = (timeStr) => {
  if (timeStr === 'Щойно') return 0;

  const match = timeStr.match(/(\d+)\s+(.+)\s+тому/);
  if (!match) return 999;

  const num = parseInt(match[1]);
  const unit = match[2];

  if (unit.includes('хвилин')) return num;
  if (unit.includes('годин')) return num * 60;
  if (unit.includes('д')) return num * 60 * 24;
  return 999;
};

return (
  <div>
    {error && <Alert message={error} type="error" />}

    {loading ? (
      <div className="loading-container">
        <LoadingSpinner size="large" text="Завантаження даних..." />
      </div>
    ) : (
      <>
        <div className="dashboard-header-bottom dashboard-card">
          <h2>Панель керування бібліотекою</h2>
          {lastUpdated && (
            <p className="last-updated">
              Останнє оновлення: {lastUpdated.toLocaleTimeString('uk-UA')}
            </p>
          )}
        </div>
```

```
      <div className="stats-cards-grid">
        <div className="dashboard-card stat-card">
          <div className="stat-icon">
          <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="currentColor" className="bi bi-person"
viewBox="0 0 16 16">
          <path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6m2-3a2 2 0 1 1-4 0 2 2 0 0 1 4 0m4 8c0 1-1 1-1 1H3s-1 0-1-1 1-4 6-4 6 3 6 4m-1-
.004c-.001-.246-.154-.986-.832-1.664C11.516 10.68 10.289 10 8 10s-3.516.68-4.168 1.332c-.678.678-.83 1.418-.832 1.664z"/>
          </svg>
          </div>
          <div className="stat-content">
            <h3 className="stat-title">Всього користувачів</h3>
            <p className="stat-value">{stats.totalUsers}</p>
          </div>
        </div>

        <div className="dashboard-card stat-card">
          <div className="stat-icon">
          <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="currentColor" className="bi bi-book" viewBox="0
0 16 16">
          <path d="M1 2.828c.885-.37 2.154-.769 3.388-.893 1.33-.134 2.458.063 3.112.752v9.746c-.935-.53-2.12-.603-3.213-.493-
1.18.12-2.37.461-3.287.811zm7.5-.141c.654-.689 1.782-.886 3.112-.752 1.234.124 2.503.523 3.388.893v9.923c-.918-.35-2.107-
.692-3.287-.81-1.094-.111-2.278-.039-3.213.492zM8 1.783C7.015.936 5.587.81 4.287.94c-1.514.153-3.042.672-3.994 1.105A.5.5 0
0 0 0 2.5v11a.5.5 0 0 0 .707.455c.882-.4 2.303-.881 3.68-1.02 1.409-.142 2.59.087 3.223.877a.5.5 0 0 0 .78 0c.633-.79 1.814-1.019
3.222-.877 1.378.139 2.8.62 3.681 1.02A.5.5 0 0 0 16 13.5v-11a.5.5 0 0 0-.293-.455c-.952-.433-2.48-.952-3.994-1.105C10.413.809
8.985.936 8 1.783"/>
          </svg>
          </div>
          <div className="stat-content">
            <h3 className="stat-title">Всього книг</h3>
            <p className="stat-value">{stats.totalBooks}</p>
            <p className="stat-subtitle">Позичень: {stats.totalBorrows}</p>
          </div>
        </div>

        <div className="dashboard-card stat-card">
          <div className="stat-icon">
          <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="currentColor" className="bi bi-file-earmark-bar-
graph" viewBox="0 0 16 16"><path d="M10 13.5a.5.5 0 0 0 .5.5h1a.5.5 0 0 0 .5-.5v-6a.5.5 0 0 0-.5-.5h-1a.5.5 0 0 0-.5.5zm-2.5.5a.5.5
0 0 1-.5-.5v-4a.5.5 0 0 1 .5-.5h1a.5.5 0 0 1 .5.5v4a.5.5 0 0 1-.5.5zm-3 0a.5.5 0 0 1-.5-.5v-2a.5.5 0 0 1 .5-.5h1a.5.5 0 0 1 .5.5v2a.5.5 0 0
```

```
1-.5.5z"/><path d="M14 14V4.5L9.5 0H4a2 2 0 0 0-2 2v12a2 2 0 0 0 2 2h8a2 2 0 0 0 2-2M9.5 3A1.5 1.5 0 0 0 11 4.5h2V14a1 1 0 0 1-1 1H4a1 1 0 0 1-1-1V2a1 1 0 0 1 1-1h5.5z"/>
          </svg>
          </div>
          <div className="stat-content">
            <h3 className="stat-title">Всього файлів</h3>
            <p className="stat-value">{stats.totalFiles}</p>
            <p className="stat-subtitle">Звітів: {stats.recentReports.length}</p>
          </div>
        </div>
      </div>

      <div className="dashboard-grid">
        <div className="dashboard-card latest-data-card">
          <h3 className="card-title">Останні завантажені дані</h3>
          {stats.latestFile ? (
            <div className="latest-file-info">
              <div className="file-header">
                <div className="file-icon">
                  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
                    <path d="M5 4a.5.5 0 0 0 0 1h6a.5.5 0 0 0 0-1H5zm-.5 2.5A.5.5 0 0 1 5 6h6a.5.5 0 0 1 0 1H5a.5.5 0 0 1-.5-.5zM5 8a.5.5 0 0 0 0 1h6a.5.5 0 0 0 0-1H5zm0 2a.5.5 0 0 0 0 1h3a.5.5 0 0 0 0-1H5z"/>
                    <path d="M2 2a2 2 0 0 1 2-2h8a2 2 0 0 1 2 2v12a2 2 0 0 1-2 2H4a2 2 0 0 1-2-2V2zm10-1H4a1 1 0 0 0-1 1v12a1 1 0 0 0 1 1h8a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1z"/>
                  </svg>
                </div>
                <div className="file-name">{stats.latestFile.filename}</div>
              </div>
              <div className="file-details">
                <div className="file-detail-item">
                  <span className="detail-label">Дата завантаження:</span>
                  <span className="detail-value">{new Date(stats.latestFile.upload_date).toLocaleDateString('uk-UA')}</span>
                </div>
                <div className="file-detail-item">
                  <span className="detail-label">Завантажив:</span>
                  <span className="detail-value">{stats.latestFile.uploader_email || "Користувач системи"}</span>
                </div>
                <div className="file-detail-item">
                  <span className="detail-label">Статус:</span>
```

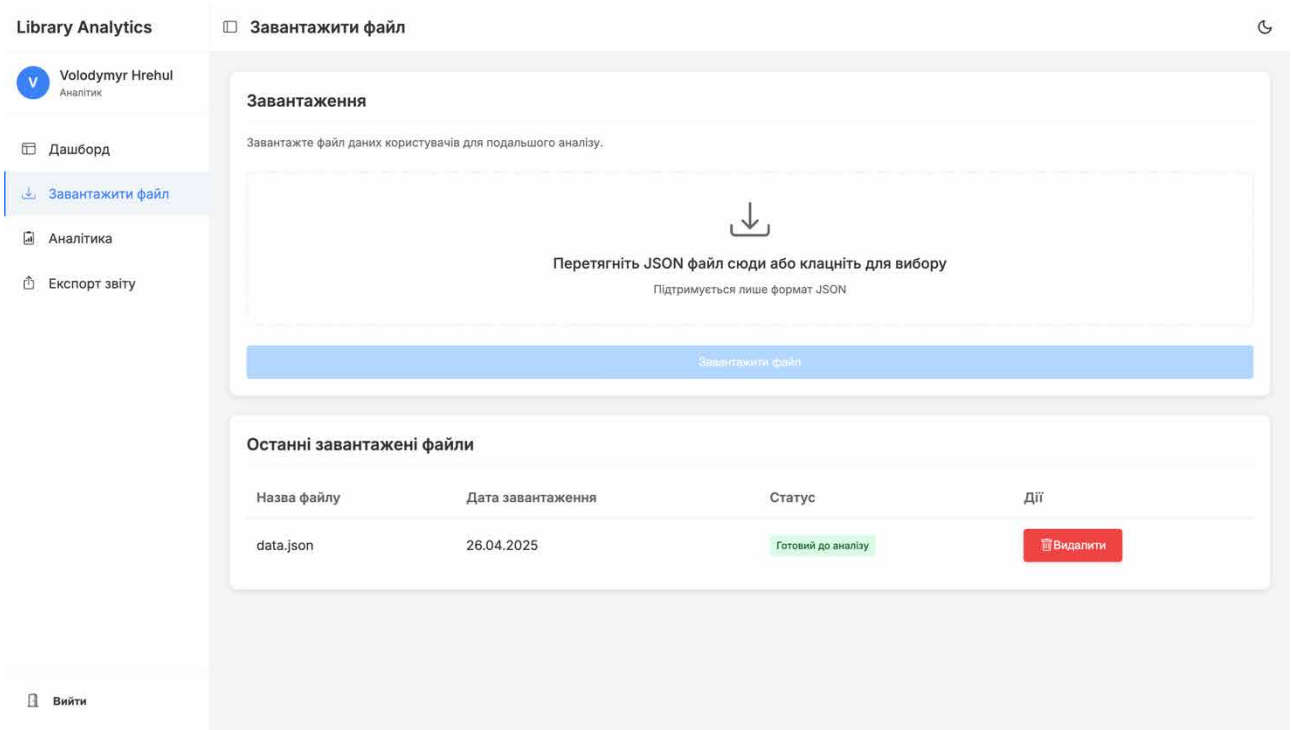
```
            <span className="detail-value status-ready">Готовий до аналізу</span>
          </div>
        </div>
        <div className="file-actions">
          <Link to="/dashboard/analytics" className="action-button primary">
            Аналізувати дані
          </Link>
          <Link to="/dashboard/upload" className="action-button secondary">
            Завантажити новий файл
          </Link>
        </div>
      </div>
    ) : (
      <div className="empty-state">
        <p className="empty-state-p">Немає завантажених файлів даних</p>
        <Link to="/dashboard/upload" className="action-button primary">
          Завантажити файл
        </Link>
      </div>
    )}
  </div>

  <div className="dashboard-card popular-genres-card">
    <h3 className="card-title">Популярні жанри</h3>
    {stats.popularGenres.length > 0 ? (
      <div className="genres-chart">
        {stats.popularGenres.map((genre, index) => {
          const maxCount = Math.max(...stats.popularGenres.map(g => g.count));
          const percentage = (genre.count / maxCount) * 100;

          return (
            <div key={index} className="genre-bar">
              <span className="genre-name">{genre.name}</span>
              <div className="genre-bar-container">
                <div
                  className="genre-bar-fill"
                  style={{ width: `${percentage}%` }}
                />
              </div>
```

```
            <span className="genre-count">{genre.count}</span>
          </div>
        );
      })}
    </div>
  ) : (
    <div className="empty-state">
      <p className="empty-state-p">Немає даних про жанри</p>
    </div>
  )}
 </div>
</div>

<div className="dashboard-grid">
 <div className="dashboard-card">
  <h3 className="card-title">Популярні книги</h3>
  {stats.topBooks.length > 0 ? (
    <div className="top-books-list">
      {stats.topBooks.map((book, index) => (
        <div key={index} className="top-book-item">
          <div className="book-rank">{index + 1}</div>
          <div className="book-title">{book.title}</div>
          <div className="book-borrows">{book.count} позичень</div>
        </div>
      ))}
    </div>
  ) : (
    <div className="empty-state">
      <p className="empty-state-p">Немає даних про книги</p>
    </div>
  )}
 </div>

 <div className="dashboard-card">
  <h3 className="card-title">Останні звіти</h3>
  {stats.recentReports.length > 0 ? (
    <div className="reports-list">
      {stats.recentReports.map((report, index) => (
        <div key={index} className="report-item">
```

```
              <div className="report-icon">
                <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
                  <path d="M9.5 0a.5.5 0 0 1 .5.5.5.5 0 0 0 .5.5.5.5 0 0 1 .5.5V2a.5.5 0 0 1-.5.5h-5A.5.5 0 0 1 5 2v-.5a.5.5 0 0 1 .5-.5.5.5 0 0 0 .5-.5.5.5 0 0 1 .5-.5z"/>
                  <path d="M3 2.5a.5.5 0 0 1 .5-.5H4a.5.5 0 0 0 0-1h-.5A1.5 1.5 0 0 0 2 2.5v12A1.5 1.5 0 0 0 3.5 16h9a1.5 1.5 0 0 0 1.5-1.5v-12A1.5 1.5 0 0 0 12.5 1H12a.5.5 0 0 0 0 1h.5a.5.5 0 0 1 .5.5v12a.5.5 0 0 1-.5.5h-9a.5.5 0 0 1-.5-.5z"/>
                  <path d="M10 7a1 1 0 1 1 2 0v5a1 1 0 1 1-2 0zm-6 4a1 1 0 1 1 2 0v1a1 1 0 1 1-2 0zm4-3a1 1 0 0 0-1 1v3a1 1 0 1 0 2 0V9a1 1 0 0 0-1-1"/>
                </svg>
              </div>
              <div className="report-content">
                <div className="report-name">{report.report_name}</div>
                <div className="report-date">{new Date(report.created_at).toLocaleDateString('uk-UA')}</div>
              </div>
              <div className="report-actions">
                <Link to={`/dashboard/analytics?report=${report.id}`} className="report-link">
                  Переглянути
                </Link>
              </div>
            </div>
          ))}
        </div>
      ) : (
        <div className="empty-state">
          <p className="empty-state-p">Немає створених звітів</p>
          <Link to="/dashboard/analytics" className="action-button primary">
            Створити звіт
          </Link>
        </div>
      )}
    </div>
  </div>
 </>
)}

<button
  className="refresh-button"
  onClick={fetchDashboardData}
  disabled={loading}
```

```
    >
      {loading ? <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" className="bi bi-clock"
viewBox="0 0 16 16">
  <path d="M8 3.5a.5.5 0 0 0-1 0V9a.5.5 0 0 0 .252.434l3.5 2a.5.5 0 0 0 .496-.868L8 8.71z"/>
  <path d="M8 16A8 8 0 1 0 8 0a8 8 0 0 0 0 16m7-8A7 7 0 1 1 1 8a7 7 0 0 1 14 0"/>
</svg> : <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" className="bi bi-arrow-clockwise"
viewBox="0 0 16 16">
  <path fillRule="evenodd" d="M8 3a5 5 0 1 0 4.546 2.914.5.5 0 0 1 .908-.417A6 6 0 1 1 8 2z"/>
  <path d="M8 4.466V.534a.25.25 0 0 1 .41-.192l2.36 1.966c.12.1.12.284 0 .384L8.41 4.658A.25.25 0 0 1 8 4.466"/>
</svg>}
    </button>
   </div>
  );
}

export default DashboardHome;
```

Сторінка завантаження файлів:



Програмна реалізація:

UploadFile.jsx:

```
import { useState, useEffect, useRef } from "react"
import Alert from "../components/Alert"
import { fileAPI } from "../services/api"

function UploadFile() {
 const [file, setFile] = useState(null)
 const [isLoading, setIsLoading] = useState(false)
 const [alert, setAlert] = useState({ show: false, message: "", type: "" })
 const [uploadedFiles, setUploadedFiles] = useState([])
 const [isLoadingFiles, setIsLoadingFiles] = useState(false)
 const [isDragging, setIsDragging] = useState(false)
 const [isDeleting, setIsDeleting] = useState(false)
 const fileInputRef = useRef(null)

 // Fetch uploaded files when component mounts
 useEffect(() => {
  fetchUploadedFiles()
 }, [])

 const fetchUploadedFiles = async () => {
  setIsLoadingFiles(true)
  try {
   const response = await fileAPI.getUploadedFiles()
   if (response.success) {
    setUploadedFiles(response.data)
   } else {
    console.error("Failed to fetch files:", response.error)
   }
  } catch (error) {
   console.error("Error fetching files:", error)
  } finally {
   setIsLoadingFiles(false)
  }
 }

 const handleFileChange = (e) => {
  const selectedFile = e.target.files[0]
```

```
  if (selectedFile) {
    validateAndSetFile(selectedFile)
  }
}

const validateAndSetFile = (selectedFile) => {
  // Check if file is JSON
  if (!selectedFile.name.endsWith('.json')) {
    setAlert({
      show: true,
      message: "Підтримуються лише файли формату JSON",
      type: "error",
    })
    return false
  }

  setFile(selectedFile)
  setAlert({ show: false, message: "", type: "" })
  return true
}

// Drag and drop handlers
const handleDragEnter = (e) => {
  e.preventDefault()
  e.stopPropagation()
  setIsDragging(true)
}

const handleDragLeave = (e) => {
  e.preventDefault()
  e.stopPropagation()
  setIsDragging(false)
}

const handleDragOver = (e) => {
  e.preventDefault()
  e.stopPropagation()
}
```

```
const handleDrop = (e) => {
  e.preventDefault()
  e.stopPropagation()
  setIsDragging(false)

  const droppedFile = e.dataTransfer.files[0]
  if (droppedFile) {
    validateAndSetFile(droppedFile)
  }
}

const handleDropAreaClick = () => {
  fileInputRef.current.click()
}

const handleSubmit = async (e) => {
  e.preventDefault()

  if (!file) {
    setAlert({
      show: true,
      message: "Будь ласка, виберіть файл",
      type: "error",
    })
    return
  }

  setIsLoading(true)

  try {
    const response = await fileAPI.uploadFile(file)

    if (response.success) {
      setAlert({
        show: true,
        message: `Файл "${file.name}" успішно завантажено`,
        type: "success",
      })
```

```
      // Refresh the file list
      fetchUploadedFiles()

      // Reset file input
      setFile(null)
      if (fileInputRef.current) {
        fileInputRef.current.value = ""
      }
    } else {
      throw new Error(response.error)
    }
  } catch (error) {
    console.error(error)
    setAlert({
      show: true,
      message: `Помилка при завантаженні файлу: ${error.message}`,
      type: "error",
    })
  } finally {
    setIsLoading(false)
  }
}

const handleDeleteFile = async (fileId, fileName) => {
  if (window.confirm(`Ви впевнені, що хочете видалити файл "${fileName}"?`)) {
    setIsDeleting(true)
    try {
      const response = await fileAPI.deleteDataFile(fileId)

      if (response.success) {
        setAlert({
          show: true,
          message: `Файл "${fileName}" успішно видалено`,
          type: "success",
        })

        fetchUploadedFiles()
      } else {
        throw new Error(response.error)
```

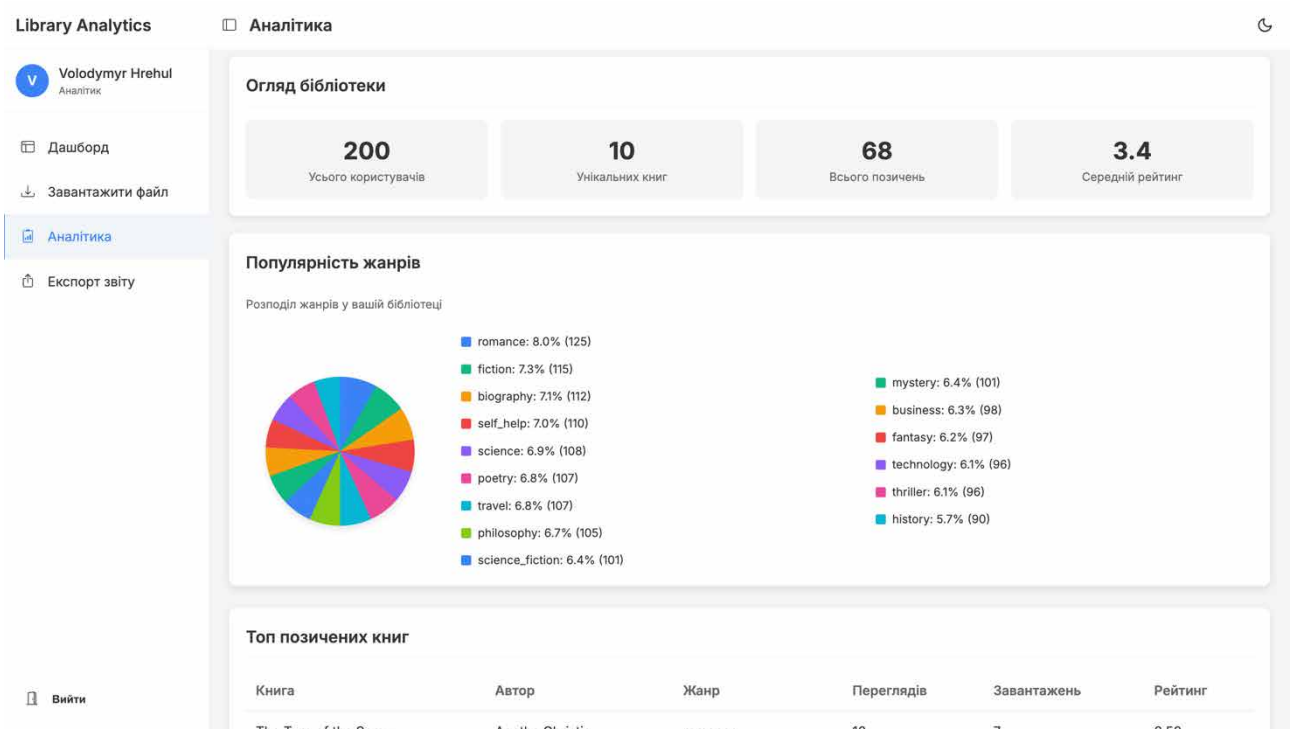
```
      }
    } catch (error) {
      console.error(error)
      setAlert({
        show: true,
        message: `Помилка при видаленні файлу: ${error.message}`,
        type: "error",
      })
    } finally {
      setIsDeleting(false)
    }
  }
}

const formatDate = (dateString) => {
  const date = new Date(dateString)
  return date.toLocaleDateString('uk-UA')
}

return (
  <div className="upload-file">
    <div className="dashboard-card">
      <h2 className="card-title">Завантаження</h2>
      <p className="card-description">Завантажте файл даних користувачів для подальшого аналізу.</p>

      <form onSubmit={handleSubmit} className="upload-form">
        <div
          className={`file-upload-container ${isDragging ? 'dragging' : ''}`}
          onDragEnter={handleDragEnter}
          onDragOver={handleDragOver}
          onDragLeave={handleDragLeave}
          onDrop={handleDrop}
          onClick={handleDropAreaClick}
        >
          <div className="file-upload-content">
            <svg xmlns="http://www.w3.org/2000/svg" width="48" height="48" fill="currentColor" className="upload-icon"
viewBox="0 0 16 16">
              <path d="M.5 9.9a.5.5 0 0 1 .5.5v2.5a1 1 0 0 0 1 1h12a1 1 0 0 0 1-1v-2.5a.5.5 0 0 1 1 0v2.5a2 2 0 0 1-2 2H2a2 2 0 0 1-2-2v-
2.5a.5.5 0 0 1 .5-.5"/>
```

```
          <path d="M7.646 11.854a.5.5 0 0 0 .708 0l3-3a.5.5 0 0 0-.708-.708L8.5 10.293V1.5a.5.5 0 0 0-1 0v8.793L5.354 8.146a.5.5 0
1 0-.708.708z"/>
        </svg>
        <p className="upload-text">
         {file
          ? file.name
          : isDragging
           ? "Відпустіть файл тут"
           : "Перетягніть JSON файл сюди або клацніть для вибору"
         }
        </p>
        <p className="upload-hint">Підтримується лише формат JSON</p>
       </div>
       <input
        ref={fileInputRef}
        id="file-upload"
        type="file"
        accept=".json"
        onChange={handleFileChange}
        className="file-upload-input"
       />
      </div>

      {file && (
       <div className="file-info">
        <p>
         <strong>Назва файлу:</strong> {file.name}
        </p>
        <p>
         <strong>Розмір:</strong> {(file.size / 1024).toFixed(2)} KB
        </p>
        <p>
         <strong>Тип:</strong> {file.type || "application/json"}
        </p>
       </div>
      )}

      {alert.show && <Alert message={alert.message} type={alert.type} />}
```

```
    <button type="submit" className="form-button" disabled={isLoading || !file}>
     {isLoading ? "Завантаження..." : "Завантажити файл"}
    </button>
   </form>
  </div>

  <div className="dashboard-card">
   <h3 className="card-title">Останні завантажені файли</h3>
   <div className="files-table-container">
    {isLoadingFiles ? (
     <p>Завантаження списку файлів...</p>
    ) : uploadedFiles.length === 0 ? (
     <p>Немає завантажених файлів</p>
    ) : (
     <table className="files-table">
      <thead>
       <tr>
        <th>Назва файлу</th>
        <th>Дата завантаження</th>
        <th>Статус</th>
        <th>Дії</th>
       </tr>
      </thead>
      <tbody>
       {uploadedFiles.map((file) => (
        <tr key={file.id}>
         <td>{file.filename}</td>
         <td>{formatDate(file.upload_date)}</td>
         <td>
          <span className="status success">Готовий до аналізу</span>
         </td>
         <td>
          <button
           className="danger-button"
           onClick={() => handleDeleteFile(file.id, file.filename)}
           disabled={isDeleting}
          >
           <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
```

```
                <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0 0-1 0v6a.5.5 0 0 0 1 0V6z"/>
                <path fillRule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4h-.5a1 1 0 0 1-1-1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
              </svg>
              Видалити
            </button>
          </td>
        </tr>
      ))}
    </tbody>
  </table>
 )}
 </div>
 </div>
 </div>
 )
}

export default UploadFile
```

Сторінка аналізу та перегляду аналітики:



Кодова частина реалізованого інтерфейсу:

Analytics.jsx:

```
import { useState, useEffect } from "react";
import GenreChart from "../components/analytics/GenreChart";
import BooksTable from "../components/analytics/BooksTable";
import UserDemographicsChart from "../components/analytics/UserDemographicsChart";
import RetentionChart from "../components/analytics/RetentionChart";
import BookCompletionChart from "../components/analytics/BookCompletionChart";
import LoadingSpinner from "../components/LoadingSpinner";
import Alert from "../components/Alert";
import { fileAPI, analysisAPI } from "../services/api";

function Analytics() {
 const [files, setFiles] = useState([]);
 const [selectedFile, setSelectedFile] = useState(null);
 const [reports, setReports] = useState([]);
 const [currentReport, setCurrentReport] = useState(null);
 const [loading, setLoading] = useState(false);
 const [isAnalyzing, setIsAnalyzing] = useState(false);
```

```
const [analysisData, setAnalysisData] = useState(null);
const [alertMessage, setAlertMessage] = useState("");
const [alertType, setAlertType] = useState("");
const [newReportName, setNewReportName] = useState("");
const [activeTab, setActiveTab] = useState("overview");

// Fetch uploaded files when component mounts
useEffect(() => {
  fetchFiles();
  fetchReports();
}, []);



const fetchFiles = async () => {
  setLoading(true);
  try {
    const response = await fileAPI.getUploadedFiles();
    if (response.success) {
      setFiles(response.data);
      if (response.data.length > 0 && !selectedFile) {
        setSelectedFile(response.data[0].id);
      }
    } else {
      showAlert(response.error, "error");
    }
  } catch (error) {
    console.error("Error fetching files:", error);
    showAlert("Не вдалося завантажити файли даних", "error");
  } finally {
    setLoading(false);
  }
};

const fetchReports = async () => {
  try {
    const response = await analysisAPI.getReports();
    if (response.success) {
      setReports(response.data);
```

```
    } else {
      console.error("Failed to fetch reports:", response.error);
    }
  } catch (error) {
    console.error("Error fetching reports:", error);
  }
};

const handleFileSelect = (e) => {
  setSelectedFile(parseInt(e.target.value));
  setAnalysisData(null);
};

const handleReportSelect = async (e) => {
  const reportId = parseInt(e.target.value);
  if (reportId) {
    setLoading(true);
    try {
      const response = await analysisAPI.getReportById(reportId);
      if (response.success) {
        setCurrentReport(response.data);
        setAnalysisData(response.data.report_data);
        setNewReportName(response.data.report_name);
      } else {
        showAlert(response.error, "error");
      }
    } catch (error) {
      console.error("Error fetching report:", error);
      showAlert("Не вдалося завантажити звіт", "error");
    } finally {
      setLoading(false);
    }
  } else {
    setCurrentReport(null);
    setAnalysisData(null);
  }
};

const handleAnalyze = async () => {
```

```
  if (!selectedFile) {
    showAlert("Будь ласка, оберіть файл для аналізу", "error");
    return;
  }
  if (!newReportName.trim()) {
    showAlert("Будь ласка, введіть назву звіту", "error");
    return;
  }

  setIsAnalyzing(true);
  try {
    const response = await analysisAPI.generateReport(selectedFile, newReportName);
    if (response.success) {
      showAlert("Аналіз успішно завершено", "success");
      await fetchReports();
      const reportData = await analysisAPI.getReportById(response.data.id);
      if (reportData.success) {
        setCurrentReport(reportData.data);
        setAnalysisData(reportData.data.report_data);
      }
    } else {
      showAlert(response.error, "error");
    }
  } catch (error) {
    console.error("Analysis error:", error);
    showAlert("Не вдалося проаналізувати дані", "error");
  } finally {
    setIsAnalyzing(false);
  }
};

const showAlert = (message, type) => {
  setAlertMessage(message);
  setAlertType(type);
  setTimeout(() => setAlertMessage(""), 5000);
};

const transformBooksData = () => {
  if (!analysisData?.content_performance) return [];
```

```
  const {
    top_borrowed_books = {},
    genre_popularity = {},
    avg_ratings_by_genre = {},
    top_authors = {},
  } = analysisData.content_performance;

  const authors = Object.keys(top_authors);
  return Object.entries(top_borrowed_books)
    .map(([title, count], index) => {
      const genres = Object.keys(genre_popularity);
      const genre = genres[index % genres.length] || "";
      const author = authors[index % authors.length] || "Невідомо";
      return {
        title,
        author,
        genre,
        views: count,
        downloads: Math.floor(count * 0.7),
        rating: avg_ratings_by_genre[genre] || 4.0,
      };
    })
    .slice(0, 10);
};

return (
  <div>
    <div className="dashboard-card">
      <h2 className="card-title">Аналітика даних користувачів</h2>
      <p className="card-description">
        Проведення аналітики даних користувачів онлайн-бібліотеки
      </p>

      {alertMessage && <Alert message={alertMessage} type={alertType} />}

      <div className="analytics-controls">
        <div className="control-group">
          <label>Виберіть файл з даними:</label>
          <select
```

```
        onChange={handleFileSelect}
        value={selectedFile || ""}
        disabled={loading || isAnalyzing}
      >
        <option value="">-- Виберіть файл --</option>
        {files.map(file => (
          <option key={file.id} value={file.id}>
            {file.filename} (
            {new Date(file.upload_date).toLocaleDateString('uk-UA')})
          </option>
        ))}
      </select>
    </div>

    <div className="control-group">
      <label>Назва звіту:</label>
      <input
        type="text"
        value={newReportName}
        onChange={e => setNewReportName(e.target.value)}
        placeholder="Введіть назву звіту"
        disabled={loading || isAnalyzing}
      />
    </div>

    <div className="control-button-group">
      <button
        className="analyze-button"
        onClick={handleAnalyze}
        disabled={!selectedFile || isAnalyzing || !newReportName.trim()}
      >
        {isAnalyzing ? "Аналізую..." : "Аналізувати дані"}
      </button>
    </div>

    <div className="control-group">
      <label>Збережені звіти:</label>
      <select
        onChange={handleReportSelect}
```

```
        disabled={loading || isAnalyzing}
      >
        <option value="">-- Виберіть звіт --</option>
        {reports.map(report => (
          <option key={report.id} value={report.id}>
            {report.report_name} (
            {new Date(report.created_at).toLocaleDateString('uk-UA')})
          </option>
        ))}
      </select>
    </div>
  </div>

  {analysisData && (
    <div className="analytics-tabs">
      <button
        className={`tab-button ${activeTab === 'overview' ? 'active' : ''}`}
        onClick={() => setActiveTab('overview')}
      >
        Огляд
      </button>
      <button
        className={`tab-button ${activeTab === 'users' ? 'active' : ''}`}
        onClick={() => setActiveTab('users')}
      >
        Аналіз користувачів
      </button>
      <button
        className={`tab-button ${activeTab === 'content' ? 'active' : ''}`}
        onClick={() => setActiveTab('content')}
      >
        Аналіз контенту
      </button>
    </div>
  )}
</div>

{loading ? (
  <div className="loading-container">
```

```
    <LoadingSpinner size="large" text="Завантаження даних..." />
  </div>
) : isAnalyzing ? (
  <div className="loading-container">
    <LoadingSpinner size="large" text="Аналіз даних..." />
  </div>
) : analysisData ? (
  <>
    {/* Огляд */}
    {activeTab === 'overview' && (
      <>
        <div className="dashboard-card">
          <h3 className="card-title">Огляд бібліотеки</h3>
          <div className="stats-highlights">
            <div className="stat-highlight-item">
              <div className="stat-value">{analysisData.total_users}</div>
              <div className="stat-label">Усього користувачів</div>
            </div>
            <div className="stat-highlight-item">
              <div className="stat-value">
                {Object.keys(analysisData.content_performance.top_borrowed_books || {}).length}
              </div>
              <div className="stat-label">Унікальних книг</div>
            </div>
            <div className="stat-highlight-item">
              <div className="stat-value">
                {Object.values(analysisData.content_performance.top_borrowed_books || {}).reduce((a, b) => a + b, 0)}
              </div>
              <div className="stat-label">Всього позичень</div>
            </div>
            <div className="stat-highlight-item">
              <div className="stat-value">
                {Object.values(analysisData.content_performance.avg_ratings_by_genre || {}).length > 0
                  ? (
                      Object.values(analysisData.content_performance.avg_ratings_by_genre)
                        .reduce((a, b) => a + b, 0) /
                      Object.values(analysisData.content_performance.avg_ratings_by_genre).length
                    ).toFixed(1)
                  : "N/A"
```

```
            }
          </div>
          <div className="stat-label">Середній рейтинг</div>
        </div>
      </div>
    </div>

    <div className="dashboard-card">
      <h3 className="card-title">Популярність жанрів</h3>
      <p className="card-description">
        Розподіл жанрів у вашій бібліотеці
      </p>
      <GenreChart genrePopularity={analysisData.content_performance.genre_popularity} />
    </div>

    <div className="dashboard-card">
      <h3 className="card-title">Топ позичених книг</h3>
      <BooksTable data={transformBooksData()} />
    </div>
  </>
)}

{/* Аналіз користувачів */}
{activeTab === 'users' && (
  <>
    <div className="dashboard-card">
      <h3 className="card-title">Демографія користувачів</h3>
      <UserDemographicsChart
        ageDistribution={analysisData.user_segments.age_distribution}
        educationDistribution={analysisData.user_segments.education_distribution}
      />
      <div className="stats-grid">
        <div className="stat-item">
          <h4>Найпопулярніші професії</h4>
          <ul>
            {analysisData.user_segments.top_professions &&
              Object.entries(analysisData.user_segments.top_professions).map(
                ([profession, count]) => (
                  <li key={profession}>
```

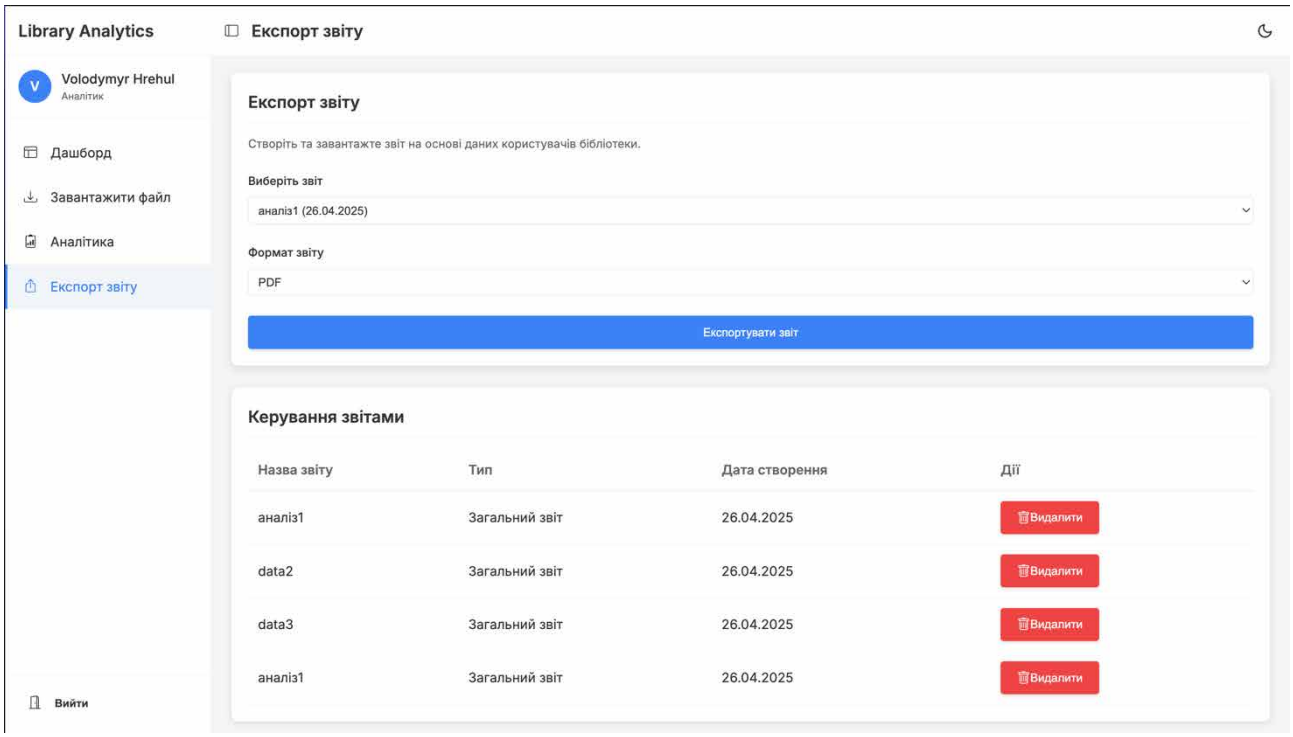
```
                <span>{profession}</span>
                <span>{count} користувачів</span>
              </li>
            )
          )}
        </ul>
      </div>
      <div className="stat-item">
        <h4>Типи акаунтів</h4>
        <ul>
          {analysisData.user_segments.account_type_distribution &&
            Object.entries(analysisData.user_segments.account_type_distribution).map(
              ([type, count]) => (
                <li key={type}>
                  <span>{type}</span>
                  <span>{count} користувачів</span>
                </li>
              )
            )}
        </ul>
      </div>
    </div>
  </div>

  <div className="dashboard-card">
    <h3 className="card-title">Утримання та активність користувачів</h3>
    <RetentionChart
      userTenure={analysisData.retention_metrics.user_tenure_distribution}
      activityByTenure={analysisData.retention_metrics.avg_activity_by_tenure}
    />
  </div>

  <div className="dashboard-card">
    <h3 className="card-title">Використання пристроїв</h3>
    <div className="stats-grid">
      <div className="stat-item">
        <h4>Середня тривалість сесії</h4>
        <ul>
          {analysisData.usage_patterns.avg_duration_by_device &&
```

```
            Object.entries(analysisData.usage_patterns.avg_duration_by_device).map(
             ([device, duration]) => (
              <li key={device}>
               <span>{device}</span>
               <span>{duration.toFixed(1)} хв.</span>
              </li>
             )
            )}
          </ul>
         </div>
         <div className="stat-item">
          <h4>Середня кількість сторінок</h4>
          <ul>
           {analysisData.usage_patterns.avg_pages_by_device &&
            Object.entries(analysisData.usage_patterns.avg_pages_by_device).map(
             ([device, pages]) => (
              <li key={device}>
               <span>{device}</span>
               <span>{pages.toFixed(0)} стор.</span>
              </li>
             )
            )}
          </ul>
         </div>
        </div>
       </div>
      </>
     )}

     {/* Аналіз контенту */}
     {activeTab === 'content' && (
      <>
       <div className="dashboard-card">
        <h3 className="card-title">Продуктивність контенту</h3>
        <div className="charts-grid">
         <div className="chart-container">
          <h4>Найпопулярніші автори</h4>
          <div className="ranking-list">
           {analysisData.content_performance.top_authors &&
```

```
          Object.entries(analysisData.content_performance.top_authors)
            .slice(0, 10)
            .map(([author, count], index) => (
              <div key={author} className="ranking-item">
                <span className="rank">{index + 1}</span>
                <span className="name">{author}</span>
                <span className="value">{count} позичень</span>
              </div>
            ))}
        </div>
      </div>
    </div>
  </div>

  <BookCompletionChart
    completionRates={analysisData.content_performance.completion_rates}
  />

  <div className="dashboard-card">
    <h3 className="card-title">Рейтинги контенту за жанрами</h3>
    <div className="ratings-chart">
      {analysisData.content_performance.avg_ratings_by_genre &&
        Object.entries(analysisData.content_performance.avg_ratings_by_genre).map(
          ([genre, rating]) => (
            <div key={genre} className="rating-bar">
              <span className="genre-name">{genre}</span>
              <div className="rating-bar-container">
                <div
                  className="rating-bar-fill"
                  style={{ width: `${(rating / 5) * 100}%` }}
                />
              </div>
              <span className="rating-value">
                {rating.toFixed(1)}
              </span>
            </div>
          )
        )}
    </div>
```

```
          </div>
        </>
      )}
    </>
  ) : (
    <div className="empty-state">
      <p>
        Оберіть файл даних та створіть звіт, щоб переглянути аналітику
      </p>
    </div>
  )}
 </div>
 );
}

export default Analytics;
```

Сторінка експорту звіту:



Програмна реалізація:

ExportReport.jsx:

```jsx
import { useState, useEffect } from "react";
import Alert from "../components/Alert";
import { analysisAPI } from "../services/api";
import LoadingSpinner from "../components/LoadingSpinner";

function ExportReport() {
 const [format, setFormat] = useState("pdf");
 const [isLoading, setIsLoading] = useState(false);
 const [alert, setAlert] = useState({ show: false, message: "", type: "" });
 const [reports, setReports] = useState([]);
 const [selectedReport, setSelectedReport] = useState("");
 const [exports, setExports] = useState([]);
 const [loadingExports, setLoadingExports] = useState(false);
 const [isDeleting, setIsDeleting] = useState(false);

 useEffect(() => {
  fetchReports();
  fetchExports();
 }, []);

 const fetchReports = async () => {
  try {
   const response = await analysisAPI.getReports();
   if (response.success) {
    setReports(response.data);
    if (response.data.length > 0) {
     setSelectedReport(response.data[0].id);
    }
   } else {
    console.error("Failed to fetch reports:", response.error);
   }
  } catch (error) {
   console.error("Error fetching reports:", error);
  }
 };

 const fetchExports = async () => {
```

```
  setLoadingExports(true);
  try {
    const response = await analysisAPI.getReportExports();
    if (response.success) {
      setExports(response.data);
    } else {
      console.error("Failed to fetch exports:", response.error);
    }
  } catch (error) {
    console.error("Error fetching exports:", error);
  } finally {
    setLoadingExports(false);
  }
};

const handleSubmit = async (e) => {
  e.preventDefault();

  if (!selectedReport) {
    setAlert({
      show: true,
      message: "Будь ласка, виберіть звіт для експорту",
      type: "error",
    });
    return;
  }

  setIsLoading(true);

  try {
    const response = await analysisAPI.exportReport(selectedReport, format);

    if (response.success) {
      setAlert({
        show: true,
        message: `Звіт успішно експортовано у форматі ${format.toUpperCase()}`,
        type: "success",
      });
```

```
      // Refresh exports list
      await fetchExports();
    } else {
      setAlert({
        show: true,
        message: response.error || "Помилка при експорті звіту",
        type: "error",
      });
    }
  } catch (error) {
    console.error(error);
    setAlert({
      show: true,
      message: "Помилка при експорті звіту",
      type: "error",
    });
  } finally {
    setIsLoading(false);
  }
};

const handleDownload = async (exportId) => {
  const exportItem = exports.find(item => item.id === exportId);
  if (!exportItem) return;

  try {
    await analysisAPI.exportReport(exportItem.report_id, exportItem.export_format);
  } catch (error) {
    console.error("Error downloading export:", error);
    setAlert({
      show: true,
      message: "Помилка при завантаженні звіту",
      type: "error",
    });
  }
};

const handleDeleteExport = async (exportId) => {
  if (window.confirm("Ви впевнені, що хочете видалити цей експорт?")) {
```

```
    setIsDeleting(true);
    try {
      const response = await analysisAPI.deleteExport(exportId);

      if (response.success) {
        setAlert({
          show: true,
          message: "Експорт успішно видалено",
          type: "success",
        });

        await fetchExports();
      } else {
        throw new Error(response.error);
      }
    } catch (error) {
      console.error("Error deleting export:", error);
      setAlert({
        show: true,
        message: "Помилка при видаленні експорту",
        type: "error",
      });
    } finally {
      setIsDeleting(false);
    }
  }
};

const handleDeleteReport = async (reportId) => {
  if (window.confirm("Ви впевнені, що хочете видалити цей звіт? Всі пов'язані експорти також будуть видалені.")) {
    setIsDeleting(true);
    try {
      const response = await analysisAPI.deleteReport(reportId);

      if (response.success) {
        setAlert({
          show: true,
          message: "Звіт успішно видалено",
          type: "success",
```

```
      });

      await fetchReports();
      await fetchExports();
     } else {
      throw new Error(response.error);
     }
    } catch (error) {
     console.error("Error deleting report:", error);
     setAlert({
      show: true,
      message: "Помилка при видаленні звіту",
      type: "error",
     });
    } finally {
     setIsDeleting(false);
    }
   }
};

return (
 <div className="export-report">
  <div className="dashboard-card">
   <h2 className="card-title">Експорт звіту</h2>
   <p className="card-description">Створіть та завантажте звіт на основі даних користувачів бібліотеки.</p>

   <form onSubmit={handleSubmit} className="report-form">
    <div className="form-group">
     <label htmlFor="report-select" className="form-label">
      Виберіть звіт
     </label>
     <select
      id="report-select"
      className="form-select"
      value={selectedReport}
      onChange={(e) => setSelectedReport(e.target.value)}
     >
      <option value="">-- Виберіть звіт --</option>
      {reports.map(report => (
```

```
          <option key={report.id} value={report.id}>
            {report.report_name} ({new Date(report.created_at).toLocaleDateString('uk-UA')})
          </option>
        ))}
      </select>
    </div>

    <br />

    <div className="form-group">
      <label htmlFor="format" className="form-label">
        Формат звіту
      </label>
      <select id="format" className="form-select" value={format} onChange={(e) => setFormat(e.target.value)}>
        <option value="pdf">PDF</option>
        <option value="csv">CSV</option>
        <option value="json">JSON</option>
        <option value="xlsx">Excel (XLSX)</option>
      </select>
    </div>

    {alert.show && <Alert message={alert.message} type={alert.type} />}

    <br />

    <button type="submit" className="form-button" disabled={isLoading || !selectedReport}>
      {isLoading ? "Експорт звіту..." : "Експортувати звіт"}
    </button>
  </form>
</div>

<div className="dashboard-card">
  <h3 className="card-title">Керування звітами</h3>
  {reports.length > 0 ? (
    <div className="reports-table-container">
      <table className="reports-table">
        <thead>
          <tr>
            <th>Назва звіту</th>
```

```
            <th>Тип</th>
            <th>Дата створення</th>
            <th>Дії</th>
          </tr>
        </thead>
        <tbody>
          {reports.map(report => (
            <tr key={report.id}>
              <td>{report.report_name}</td>
              <td>{getReportTypeName(report)}</td>
              <td>{new Date(report.created_at).toLocaleDateString('uk-UA')}</td>
              <td>
                <button
                  className="danger-button"
                  onClick={() => handleDeleteReport(report.id)}
                  disabled={isDeleting}
                >
                  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
                    <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0 0-1 0v6a.5.5 0 0 0 1 0V6z"/>
                    <path fillRule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4h-.5a1 1 0 0 1-1-1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
                  </svg>
                  Видалити
                </button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  ) : (
    <div className="empty-state">
      <p>Немає звітів</p>
    </div>
  )}
</div>
```

```
    <div className="dashboard-card">
     <h3 className="card-title">Останні експортовані звіти</h3>

     {loadingExports ? (
      <div className="loading-container">
       <LoadingSpinner size="medium" text="Завантаження звітів..." />
      </div>
     ) : exports.length > 0 ? (
      <div className="reports-table-container">
       <table className="reports-table">
        <thead>
         <tr>
          <th>Назва звіту</th>
          <th>Тип</th>
          <th>Дата створення</th>
          <th>Формат</th>
          <th>Дії</th>
         </tr>
        </thead>
        <tbody>
         {exports.map(exportItem => {
          const report = reports.find(r => r.id === exportItem.report_id);
          return (
           <tr key={exportItem.id}>
            <td>{report ? report.report_name : 'Невідомий звіт'}</td>
            <td>{report ? getReportTypeName(report) : '-'}</td>
            <td>{new Date(exportItem.created_at).toLocaleDateString('uk-UA')}</td>
            <td>{exportItem.export_format.toUpperCase()}</td>
            <td>
             <div className="button-group">
              <button
               className="action-button"
               onClick={() => handleDownload(exportItem.id)}
              >
               <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
                <path d="M.5 9.9a.5.5 0 0 1 .5.5v2.5a1 1 0 0 0 1 1h12a1 1 0 0 0 1-1v-2.5a.5.5 0 0 1 1 0v2.5a2 2 0 0 1-2 2H2a2 2 0 0
1-2-2v-2.5a.5.5 0 0 1 .5-.5z"/>
                <path d="M7.646 11.854a.5.5 0 0 0 .708 0l3-3a.5.5 0 0 0-.708-.708L8.5 10.293V1.5a.5.5 0 0 0-1 0v8.793L5.354
8.146a.5.5 0 1 0-.708.708l3 3z"/>
```

```
                    </svg>
                    Завантажити
                  </button>
                  <button
                    className="danger-button"
                    onClick={() => handleDeleteExport(exportItem.id)}
                    disabled={isDeleting}
                  >
                    <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" viewBox="0 0 16 16">
                      <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0 0-1 0v6a.5.5 0 0 0 1 0V6z"/>
                      <path fillRule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4h-.5a1 1 0 0 1-1-1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
                    </svg>
                    Видалити
                  </button>
                </div>
              </td>
            </tr>
          );
        })}
      </tbody>
    </table>
  </div>
) : (
  <div className="empty-state">
    <p>Немає експортованих звітів</p>
  </div>
)}
</div>
</div>
);
}

function getReportTypeName(report) {

if (!report.report_data) return 'Загальний звіт';
```

```
  if (report.report_data.usage_patterns && Object.keys(report.report_data.usage_patterns).length > 0) {
    return 'Активність користувачів';
  }

  if (report.report_data.content_performance && Object.keys(report.report_data.content_performance).length > 0) {
    return 'Популярні книги';
  }

  if (report.report_data.user_segments && Object.keys(report.report_data.user_segments).length > 0) {
    return 'Демографія користувачів';
  }

  return 'Загальний звіт';
}

export default ExportReport;
```