ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА WEB-ЗАСТОСУНКУ ПЛАНУВАЛЬНИКА КНИГ "BOOK PLANNER" МОВОЮ ПРОГРАМУВАННЯ JAVASCRIPT»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дар'я БІЛАН
(підпис)

Виконав: здобувачка вищої освіти групи ПД-43

Дар'я БІЛАН

Керівник: Віктор ГРЕБЕНЮК
доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Білан Дар'ї Борисівни

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку планувальника книг "Book planner" мовою програмування JavaScript»
керівник кваліфікаційної роботи доктор філософії (PhD) Віктор ГРЕБЕНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи планування книг, опис методів структуризації інформації відповідно до ідеї, створення, редагування та видалення даних відповідно до планування книги, технічна документація з описом бібліотек для реалізації web-застосунку для планування книг.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій планування книг.
2. Проектування застосунку для спрощення процесу структуризації даних для планування книги.

3. Програмна реалізація та опис функціонування застосунку для процесу планування книги.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма використання.
5. Діаграма класів.
6. Схема роботи Web-застосунку.
7. Сторінка книги, категорія Description.
8. Форма створення персонажа.
9. Відео-демонстрація роботи Web-застосунку.
10. Апробація.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02 - 06.03.2024	Виконано
2	Аналіз та дослідження існуючих аналогів	07.03 - 13.03.2024	Виконано
3	Огляд існуючих технологій планування книг	14.03 – 16.04.2024	Виконано
4	Проектування web-застосунку для процесу планування книги	17.04 – 18.04.2024	Виконано
5	Програмна реалізація web-застосунку для планування книг «Book planner»	19.04 – 26.04.2024	Виконано
6	Тестування застосунку	27.04 – 28.04.2024	Виконано
7	Оформлення роботи: вступ, висновки, реферат	29.04 - 05.05.2024	Виконано
8	Розробка демонстраційних матеріалів	06.05 - 12.05.2024	Виконано
9	Попередній захист роботи	13.05 – 31.05.2024	Виконано

Здобувачка вищої освіти _____ Дар’я Білан
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 2 табл., 34 рис., 9 джерел.

Мета роботи – спрощення процесу планування книги за допомогою web-застосунку, що написаний на мові JavaScript.

Об'єкт дослідження – процес планування книги.

Предмет дослідження – web-застосунок для планування книг.

Короткий зміст роботи: В роботі проаналізовано методи для процесу планування книг. Проаналізовано існуючі програмні засоби для планування книг: Stack Edit, Simplenote, Google Docs. Розроблено алгоритм роботи web-застосунку та програмно реалізовані ключові функціональні можливості, зокрема: створення об'єкту книги, редагування та видалення існуючої книги, введення та збереження опису книги та синопсису, створення, редагування та видалення персонажа, сцени, нотатки. Проведено функціональне та модульне тестування додатку. В роботі використано мову програмування JavaSЬIcript для створення web-застосунку, бібліотеку React.js для створення інтуїтивного користувацького інтерфейсу.

Сферою використання застосунку є процес планування книги та структуризація інформації відносно певної ідеї.

ЗМІСТ

ВСТУП

Написання книги – це складний процес, який складається з багатьох етапів. Після визначення ідеї для написання книги, необхідно якісно спланувати подальший розвиток та розгортання її в історію, що зможе не тільки зацікавити читача, але й розкрити посил, що закладений у сюжет. Планування є критично важливим процесом, оскільки дозволяє детально продумати кожного персонажа книги, сцени, що висвітлюватимуть основні моменти розвитку подій, дає чітке відображення концепції книги та спрощує її дотримання при подальшому написанні. Отже, необхідно приділити увагу саме плануванню перед реалізацією ідеї у повноцінну розповідь.

Актуальністю роботи є розробка застосунку, який допоможе структурувати інформацію відповідно до ідеї, створити персонажів та сцени, описати ідею книги та короткий опис всього сюжету, а також зберегти напрацювання у web-застосунку.

Об'єктом дослідження є процес планування книги.

Предметом дослідження є web-застосунок для планування книг.

Метою роботи є спрощення процесу планування книги за допомогою web-застосунку мовою JavaScript.

Завданням роботи є аналіз предметної галузі та опис процесу розробки додатку, включаючи формування технічного завдання, розробки зручного та зрозумілого інтерфейсу, безпосередньо додатку та процесу його тестування.

Під час дослідження, було проведено аналіз існуючих додатків, таких як: Stack Edit, Simplenote, Google Docs. На основі отриманих результатів, було вирішено розробити web-застосунок, який би дав змогу: створювати книги, до кожної книги створити: список персонажів, список сцен, опис самої книги, синопсис та нотатка. Для реалізації було використано мову програмування JavaScript та бібліотека React JS.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЗУЛІ

Книги мають великий вплив на суспільство. Вони є носіями інформації, що допомагають у навчанні та розвитку. Можуть мати розважальний зміст та бути частиною дозвілля, для відпочинку від роботи та проблем. Діти починають свій розвиток, використовуючи книги з картинками для ознайомлення зі світом та його жителями. А написані у давні часи манускрипти розкривають життя людей минулого, їх звичаї та відносини, проблеми суспільства та взаємодії держав. Отже, книги є важливою складовою нашого життя, як для збагачення знань з різних сфер, проводження досліджень, так і для проведення дозвілля, читаючи художню літературу та поринаючи у вигадані сюжети та світи.

Технологічний прогрес сьогодення дозволив збільшити варіанти поширення примірників книг. На разі люди можуть отримати доступ до будь-якої книги у паперовому вигляді з твердою або м’якою обкладинкою та в електронному, що дозволяє мати книгу прямо в телефоні, планшеті або персональному комп’ютері. Таке досягнення дозволило збільшити попит на літературу у суспільстві.

На замовлення Українського інституту книги у 2020 році Info Sapience провели дослідження «Читання в контексті медіаспоживання та життєконструювання». Загалом у ньому прийняло участь 3900 осіб, серед них 8 фокус-груп з дорослими респондентами віком від 18 до 59 років, що складають 2100 осіб та 1800 дітей віком від 6 до 17 років, що увійшли до складу 16 фокус-груп. За результатами дослідження виявлено, що від 2020 року у читачів значно виріс попит саме на україномовну літературу. Також серед усіх опитаних 8% дорослих та 13% дітей читають книги щоденно. Попит на літературу збільшився під час карантину серед 23% читачів дитячого віку. Було виявлено, що жінки читають більше ніж респонденти чоловічої статі. До того ж, найпопулярнішими жанрами серед читачів є класична література, її обрало 20% респондентів, детективи – 18%, любовні романи – 16%, фентезі та фантастика, а також сучасні романи – 15%. Менш популярними, але також згадуваними є книги з психології та саморозвитку, науково-популярна, фахова та бізнес-література.[1]

Дослідження проведене Info Sapience стало логічним продовженням проекту Ukrainian Reading and Publishing Data 2018, що представляло зріз стану видавництва книг та вподобань людей на літературу.

За результатами Ukrainian Reading and Publishing Data 2018 читання книг займало п'яте місце серед найбільш поширених варіантів проведення дозвілля. Серед опитаних 11% респондентів читають книги щодня, 17% кілька разів на тиждень, 20% декілька раз на місяць. По Україні найбільше людей читають у Центральному регіоні, менше у Західному та Південному регіонах. Було виявлено, що література більше популярна серед людей з вищою освітою та зайнятістю у сферах освіти, науки та творчих напрямах. На питання «Чи любите ви читати?» 44% респондентів відповіли, що не можуть уявити свого життя без читання. Також респондентам були задані питання стосовно значення читання у житті людини. 92% опитуваних погодились з тим, що читання сприяє саморозвитку. 82% зазначили, що воно також сприяє професійному зростанню. 79% відповіли, що це допомагає у комунікації з людьми. 77% вважають, що це допомагає відволіктись від рутинного життя та відпочити. 69% стверджують про збільшення поваги від інших людей. Отже, на 2018 рік ситуація з використанням та популярністю книг була досить суперечлива.[2]

Можемо сказати, що за останні роки читання поширилось на всі вікові групи. Для підтримки цікавості суспільства у літературі необхідно впроваджувати та видавати якісну літературу, що зможе зацікавити потенційного читача та залучить більше людей до читання, що значно вплине на освіченість та грамотність населення.

Написання книги починається з етапу планування, що є одним з найважливіших, оскільки автор розписує ідею етапами та частинами, створюючи в кінці готову роботу. Для написання книги важливими є опис самої ідею, навколо якої будується сюжет, детальне створення персонажів, що фігуруватимуть в історії, сцени – основна рушійна сила будь-якої історії. Отже для створення книги необхідно прикласти багато зусиль ще на етапі планування.

1.1 Аналіз та характеристика web-застосунків планувальника книг

Проведено аналіз для визначення ключових аспектів та функцій, що мають бути реалізовані у розробляємому додатку.

Web-застосунок – це програмне забезпечення, яке запускається через ваш web-браузер. Даний вид застосунків є динамічним та пропонує широкий спектр функцій. Зазвичай дані додатки пишуться мовами web-розробки, як HTML, CSS, JavaScript. Web-застосунки мають клієнт серверну архітектуру. Клієнтом виступає браузер, користувач відкриває або вводить посилання на web-застосунок. Браузер відправляє HTTPS запит на сервер, що містить інформацію про застосунок та контент у ньому. Після обробки запиту, формується відповідь у вигляді HTTP Response, яка містить розмітку сторінки на HTML, JavaScrip-код тощо та надсилає відповідь назад до браузера. Коли відповідь від сервера отримана, формується відображення вмісту відповіді, в якості сторінки додатку або інший вміст, в залежності від типу та сенсу запиту до сервера. Взаємодія між частинами додатку відбувається через мережу Інтернет.

Web-додатки мають ряд переваг:

- web-застосунки відкриваються через будь-який браузер: Opera, Chrome, Edge, Firefox, Safari тощо. Все що необхідно користувачу для взаємодії з додатком це доступ до Інтернету;
- користувачу не потрібно встановлювати додаткове програмне забезпечення на пристрій;
- оскільки інформація у web-застосунках зберігається на сервері, користувач може отримати доступ до застосунку в будь-який час з будь-якого пристрою підключеного до мережі Інтернет та з встановленим браузером;
- web-додатки впроваджуються, оновлюються та підтримуються дистанційно.

Отже, створення web-застосунку – це ідеальне рішення для впровадження застосунку для планування книг, оскільки користувач матиме доступ з будь-якого пристрою в будь-який час, що важливо, коли необхідно записати нову ідею.

Для вирішення проблеми з планування книги, проведено аналіз на необхідні елементи на функції, що мають бути у додатку.

Коли автор починає історію, не існує правильного чи неправильного шляху для її реалізації. Але часто можна потрапити у пастку власних ідей через обмірковування ситуацій та персонажів, що могли б існувати у історії. Для народження історії та якісного написання, необхідно реалізовувати усі ідеї, що виникають в голові в життя. З'являючись у дійсності, вони розширюються, отримують більше деталей та стають реальними, як для автора так і для майбутньої історії.

Більшість авторів починають свій шлях створення історії з думок про героїв, адже вони будуть рухати сюжет та впливати на події та інших персонажів. Головний протагоніст та антагоніст це ті навколо кого вибудовується сюжет. Їх взаємодія, епізоди, що виникають при зустрічі, причини їх майбутніх дій та наслідки вже скоєних. Для чіткої передачі історії, необхідно продумати, який характер буде у героя та що вплинуло на його або її формування у тому варіанті, який буде представлений для читача. Чим детальніше продуманий герой, тим простіше буде зрозуміти його вчинки та погляд на події. Оскільки персонажі мають важливу роль у книзі, то необхідно приділити увагу як саме користувач зможу створювати та зберігати. А отже, головним є можливість збереження імені персонажа, його ролі в книзі. Також, важливим є детальне пропрацювання зовнішності, характер та біографії героя.

Окрім персонажів, ключовими в історії є сцени, що розкриватимуть атмосферу, передаватимуть інформацію про події, розкриватимуть персонажів з різних сторін, що дозволить читачу емоційно зв’язатись з персонажем. Демонстрація важливих подій, розкриття світобудови роману, культур та соціального розвитку створеного світу. Вони розкривають історію, роблять її більш

живою та зрозумілою для сприйняття та поглинання в сюжет. Отже, для створення сцен необхідно мати змогу детально прописувати всю сцену, а також давати назву події, що описується.

Також, важливим є змога записати саму ідею книги, опис сюжету. Отже, застосунок має надавати функціонал для створення всіх вище зазначених пунктів.

Окрім створення інформації, необхідно структурувати її, відносити до певної книги, ідею тощо для якісної організації даних.

1.2 Огляд та аналіз програмних рішень для планування книг

Існують різні засоби для планування книги. Це можуть бути клаптики паперу, блокноти, де можна записати ідеї. Але все це не є актуальним у сучасному світі, що рухається у напрямку технологічного розвитку. Носити з собою блокнот, зошит чи будь який інший паперовий виріб аби мати змогу записати ідею, що прийшла у голову несподівано, коли на неї не очікуєш, дуже не зручно. Все це займає багато місця в руках, рюкзаку, сумці тощо. Завдяки розвитку сучасних технологій все це можна мати у власному телефоні, планшеті чи персональному комп'ютері, що значно економить простір та завжди є можливість швидкого доступу до ресурсу.

Розглядаючи можливі програмні рішення для планування книг, виділено декілька критеріїв, що мають бути у застосунку для зручного конспектування та майбутнього планування книги. Представимо їх у вигляді списку:

- можливість створення, редагування та видалення об'єктів книги, персонажа, сцени та нотатки;
- можливість введення опису та синопсису книги;
- наявність готових шаблонів для форм створення книги, персонажа та сцени;
- можливість структуризації інформації, віднесення їх до певної книги;

– можливість завантаження зображень та прив’язки їх до книги та персонажа.

Отже, відштовхуючись від критерій до застосунку, можемо провести аналіз існуючих рішень для планування книг та визначити їх переваги та недоліки.

Одним із додатків для вирішення даного питання є StackEdit – онлайн-редактор тексту, що дозволяє створювати, редагувати та форматувати документи. Для роботи з текстом він використовує мови розмітки Markdown та HTML. Даний редактор працює через браузери, що забезпечує доступ до нього для користувачів з будь-якими пристроями та операційними системами. Окрім цього, така доступність дозволяє користувачам не завантажувати додаткове програмне забезпечення на пристрій. Для синхронізації та збереження документів на хмарних сервісах необхідно авторизуватись за допомогою Google акаунту.

Для зручної структуризації інформації є можливість створення папок та файлів. Для планування книг це є дуже важливим, оскільки необхідно структурувати інформацію відповідно до тематики або ідеї. Таким чином, папка може виступати в якості майбутньої книги. До цієї папки можна додати файли, що будуть нести у собі інформацію про опис книги, що буде реалізовуватись. Окрім цього, можливо створити папки для файлів з персонажами книги, сценами та не до кінця сформованими ідеями. Можливість такої структуризації дозволяє ефективно зберігати та використовувати створені файли.

Робочий простір представлений на рисунку 1.1.

Перевагами використання StackEdit виступають наступні пункти:

– зручний та зрозумілий інтерфейс дає можливість легко працювати з сервісом без додаткового навчання;
– даний редактор є онлайн-інструментом, що не обмежується у використанні типом операційної системи, пристрою тощо. StackEdit запускається через будь-який браузер. Все що необхідно це доступ до Інтернету;
– можливість збереження файлів у хмарному середовищі до яких підключається редактор: Dropbox, GitHub, GitLab, Google Drive;
– можливість створення власної структури файлу завдяки мовами розмітки.

До недоліків StackEdit можна віднести:

– необхідність створювати шаблони власноруч для кожного об’єкту (герой книги, опис сцени);
– необхідність власноруч налаштовувати структуру книги;
– можливість втрати даних, які не збережені у хмарному середовищі доступ до якого є лише після прив’язки акаунту;
– відсутність можливості завантажувати зображення.

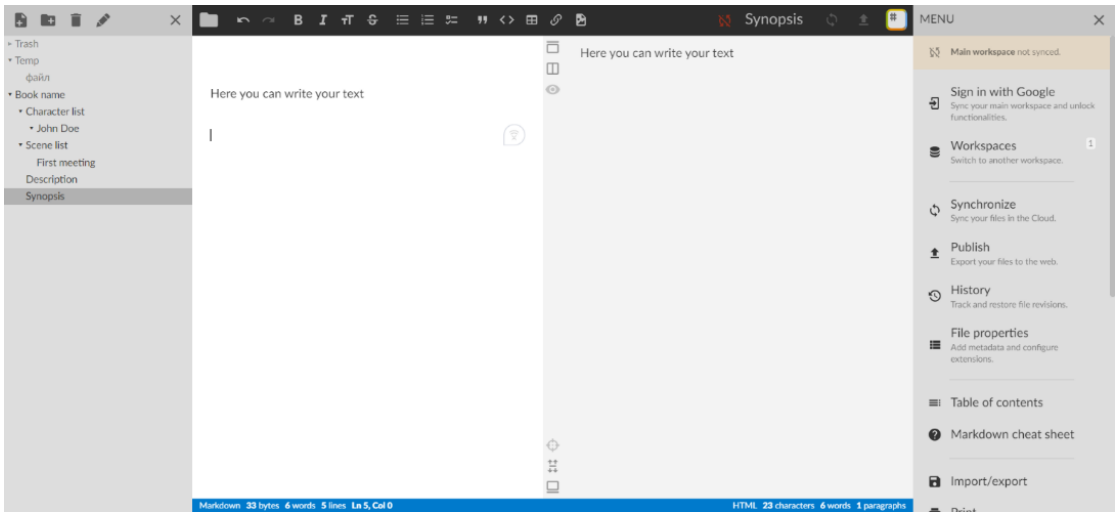


Рис. 1.1 StackEdit

Ще одним з можливих варіантів застосунків для планування книги виступає Simplenote. Це застосунок для створення нотатків, що має варіанти для платформ Windows, MacOS, Linux, Android та IOS. Цей застосунок дозволяє легко та зручно створювати дописи з будь-якого пристрою. Через наявність варіантів під різні платформи не обмежується використання додатку певним видом пристрою чи операційною системою. Для початку роботи з додатком користувач має зареєструватись, якщо до цього не користувався застосунком, або авторизуватись. Однак, даний застосунок не дає можливість структурувати нотатки для літературного планування. Ця проблема, частково, вирішується завдяки можливості додавання тегів до кожного допису, а потім проводити пошук або сортування за ними.

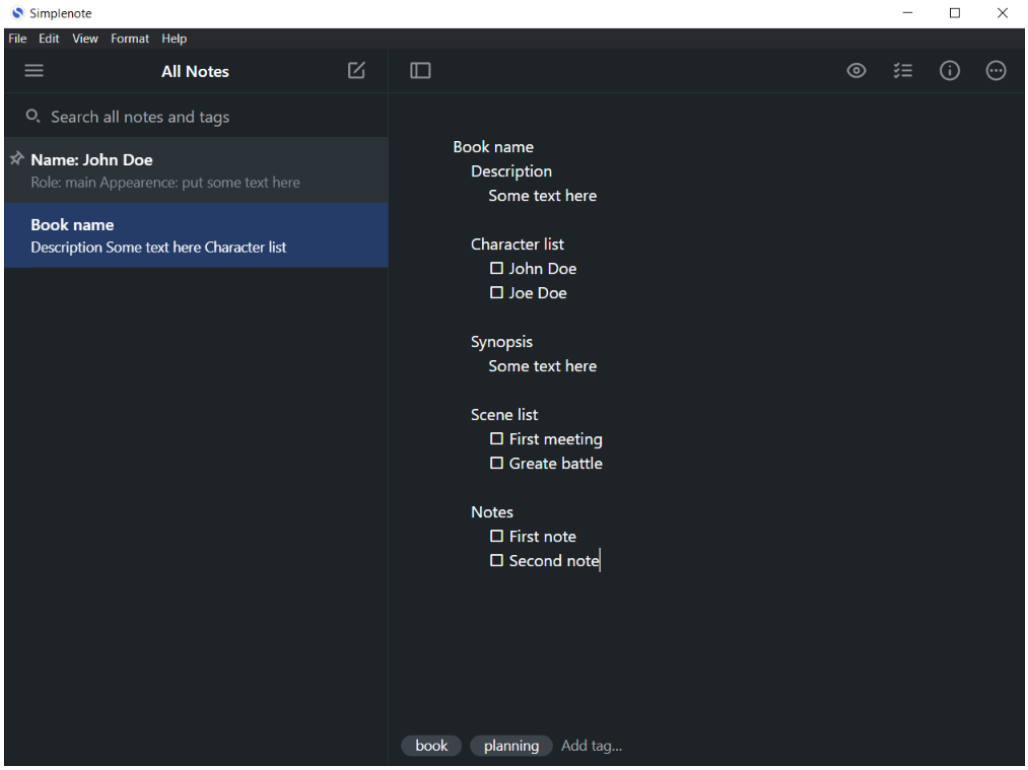


Рис. 1.2. Simplenote

Робочий простів представлений на рисунку 1.2.

Перевагами даного додатку є наступні пункти:

– інформація синхронізується між усіма пристроями з додатками, що авторизовані на один акаунт;

– організація створених нотатків та їх пошук полегшується за допомогою тегів, що можна додавати до кожного допису;
– доступність на усіх популярних платформах, таких як Windows, Linux, MacOS, Android, IOS.

Недоліками використання можна вважати:

– відсутність можливості завантажувати зображення до нотатків;
– необхідність встановлювати застосунки на пристрій;
– відсутність структури для планування книги;
– під час синхронізації між пристроями можлива втрата даних, через нестабільне підключення до Інтернету.

Google Docs – безкоштовний офісний пакет, що доступний з браузеру, створений компанією Google. Даний сервіс надає можливість створення текстових документів. Завдяки наявності текстового редактору та можливості форматування документа користувач може створювати власні шаблони для форм персонажів та опису сцен. Також, можна використовувати та підлаштовувати вже готові шаблони, що пропонуються сервісом. Google Docs не надає можливості структурувати інформацію відповідно до певної книги. Синхронізація з іншими сервісами Google дає можливість створювати таблиці або форми за необхідністю. Google Docs регулює права доступу до документів, таким чином забезпечуючи безпеку та приватність файлів.

Приклад використання Google Docs представлений у рисунку 1.3.

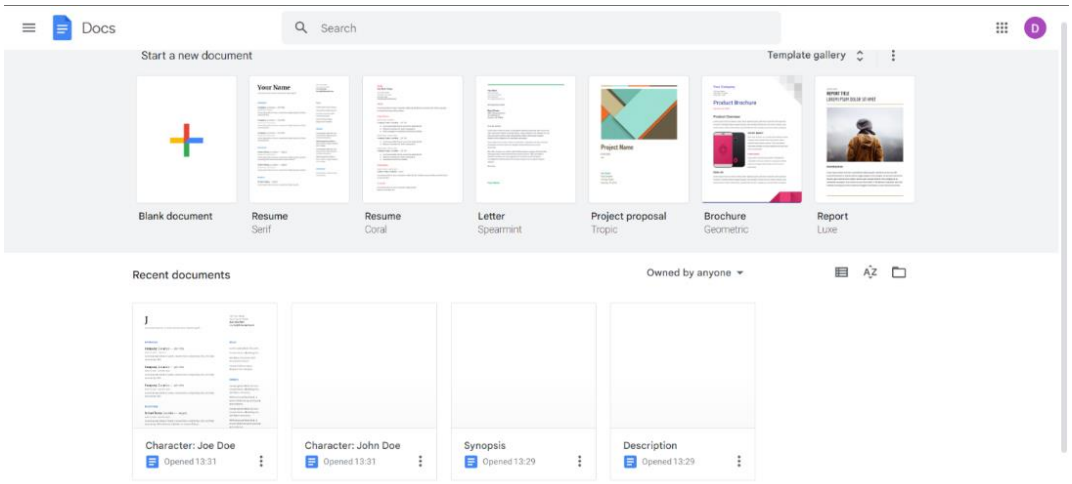


Рис. 1.3. Використання Google Docs

Перевагами використання Google Docs є:

- автоматичне збереження інформації у хмарному середовищі Google Drive;
- можливість створення власних або використання існуючих шаблонів;
- можливість завантаження зображень до файлів;
- доступність через браузер надає можливість користувачам з будь-яким типом пристрою та операційною системою використовувати даний сервіс.

Недоліками виступають наступні пункти:

- відсутність можливості автономної структуризації документів в якості дерева або вкладених файлів відповідно до структури планування книги;
- при великому об’ємі документа, інтерфейс Google Docs починає працювати із затримками;
- залежність від підключення до мережі Інтернет та можливість втрати даних при проблемах з мережею.

Враховуючі всі фактори, що були перераховані та зазначені вище, було сформовано таблицю зведених результатів аналізу характеристик застосунків-аналогів у порівнянні з розроблюваним web-застосунком, згідно зі задачею та потребами, що були поставлені для розробки програмного забезпечення. Результати наведені у таблиці. 1.1.

Таблиця 1.1.

Зведені результати аналізу характеристик додатків-аналогів

Назви аналогів **Особливості**	**Stack Edit**	**Simplenote**	**Google Docs**	**Book Planer**
Збереження інформації	кеш браузера та можуть бути втрачені	хмарне середовище Automattic	Google drive (15ГБ)	хмарне середовище Firebase Firestore (до 1 ГБ безкоштовної версії)
Структуризація інформації відповідно до певної книги	Налаштовується власноруч	Налаштовуються власноруч через додавання тегів	Налаштовується власноруч	У додатку існує готова структура для планування книги
Наявність готових шаблонів для форми створення книги	-	-	-	+
Наявність готових шаблонів для форми створення персонажу	-	-	- (можливо використання готових шаблонів для резюме)	+

Продовження таблиці 1.1

Назви аналогів **Особливості**	**Stack Edit**	**Simplenote**	**Google Docs**	**Book Planer**
Наявність готових шаблонів для форми створення сцени	-	-	-	+
Доступ через браузери	Chrome, Firefox, Opera, Safari	-	Chrome, Firefox, Opera, Safari	Chrome, Firefox, Opera, Safari
Можливість завантаження зображень	-	-	+ (окрім svg, gif, tiff)	+

1.3 Аналіз та вибір середовища розробки

Для першого етапу процесу розробки, а саме створення макету майбутнього web-застосунку, було обрано графічний онлайн-редактор Figma.

Figma – це графічний онлайн-редактор для спільної роботи. В ньому можна створювати прототипи сайтів, інтерфейси застосунків та обговорювати з колегами виправлення в режимі реального часу.

Вона була розроблена та випущена компанією Figma Inc. у 2016 році. Наразі, використовується людьми з різних напрямків, які пов’язані зі створенням дизайну,

прототипуванням застосунків, сайтів та багато різних інших проектів з візуалізації тощо.

Figma має три причини обирати її для роботи:

– Сервіс працює в будь-якому браузері та з будь-якого пристрою з будь-якою операційною системою. Тож не має необхідності встановлювати додаткові застосунки.
– Figma – онлайн-редактор, який зберігає результати роботи у хмарному середовищі та їх можуть бачити усі учасники команди. В будь-який момент можна переглянути внесені зміни та відновити резервну копію. За необхідністю.
– До кожного макету у Figma учасники можуть залишати коментарі та отримувати відповіді від колег.

Інтерфейс Figma відображений на рисунку 1.4.

Для другого етапу розробки необхідно обрати середу розробки.

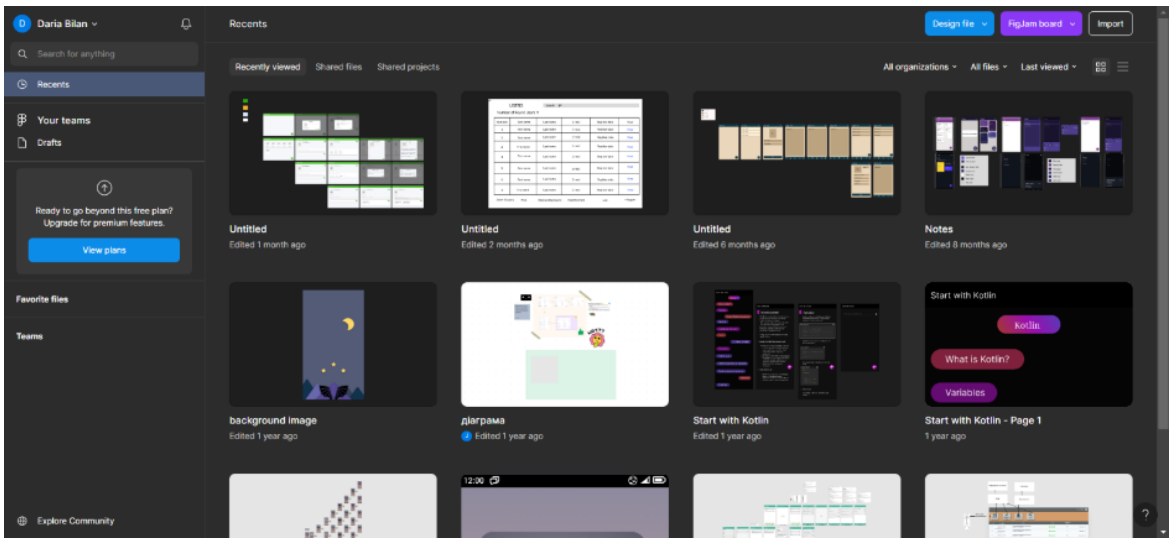


Рис.1.4. Figma

Середовище розробки або IDE (скор. англ. Integrated Development Environment – інтегроване середовище розробки) – спеціальний програмний комплекс для повного циклу написання та тестування програмного забезпечення певною мовою програмування.

Існує декілька популярних середовищ розробки для написання web-застосунків таких як: Visual Studio Code, WebStorm, Atom.

Visual Studio Code (VS Code) – один з найпопулярніших безкоштовних редакторів коду від корпорації Microsoft. VS Code поширюється та є доступним для Windows, MacOS та Linux. Має вбудовану підтримку для JavaScript, TypeScript та React.js, а також широку екосистему розширень для інших мов програмування та середовищ виконання.

Visual Studio Code надає базові функції для написання, редагування та запуску коду, рефакторинг, вбудований термінал, систему керування версіями Git, IntelliSense. За допомогою розширень надається підтримка для певних мов програмування та доповнення для редактору тощо.

Окрім цього, VS Code відомий своєю швидкістю роботи та продуктивності, особливо з великими проектами.

Інтерфейс Visual Studio Code представлений на рисунку 1.5.

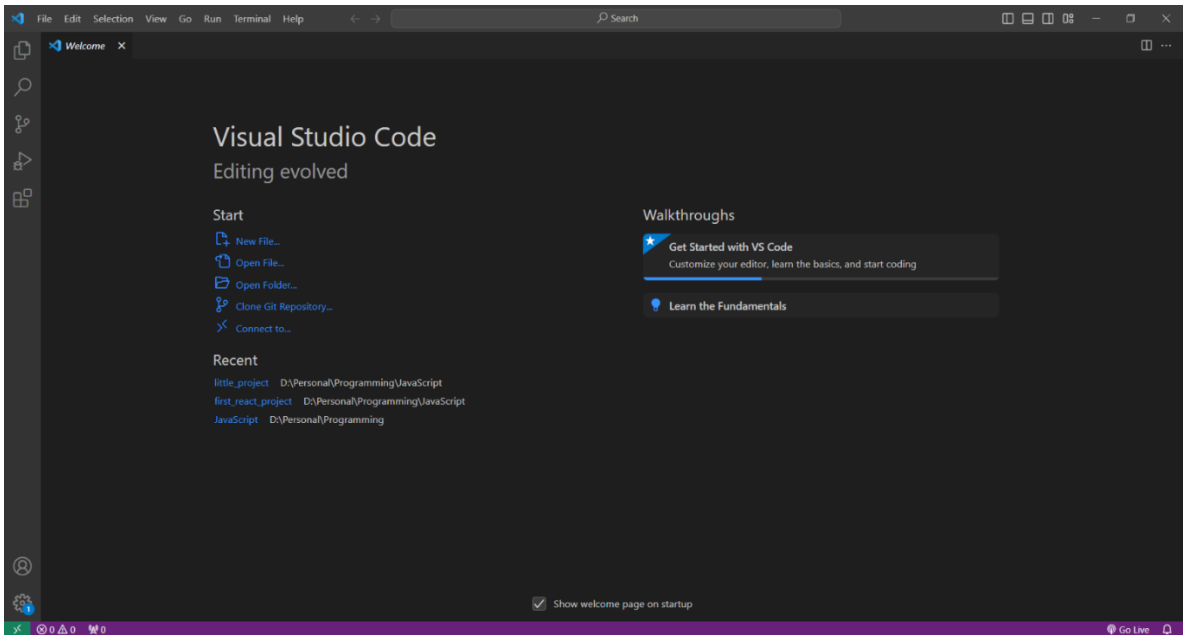


Рис.1.5. Visual Studio Code

Atom – вільний редактор коду, розроблений компанією GitHub. Atom може використовуватись як основна середа розробки, так як він має ряд можливостей для

автозаповнення, підсвічування синтаксису, підключення системи керування версіями, перевірка коду різних мов (Ruby, Python, JavaScript, C/C++ тощо).

Даний редактор має велику кількість плагінів та продовжує наповнюватись новими за допомогою спільноти розробників. Наявність плагінів дозволяє налаштувати застосунок під власні потреби.

Екран Atom представлений на рисунку 1.6.

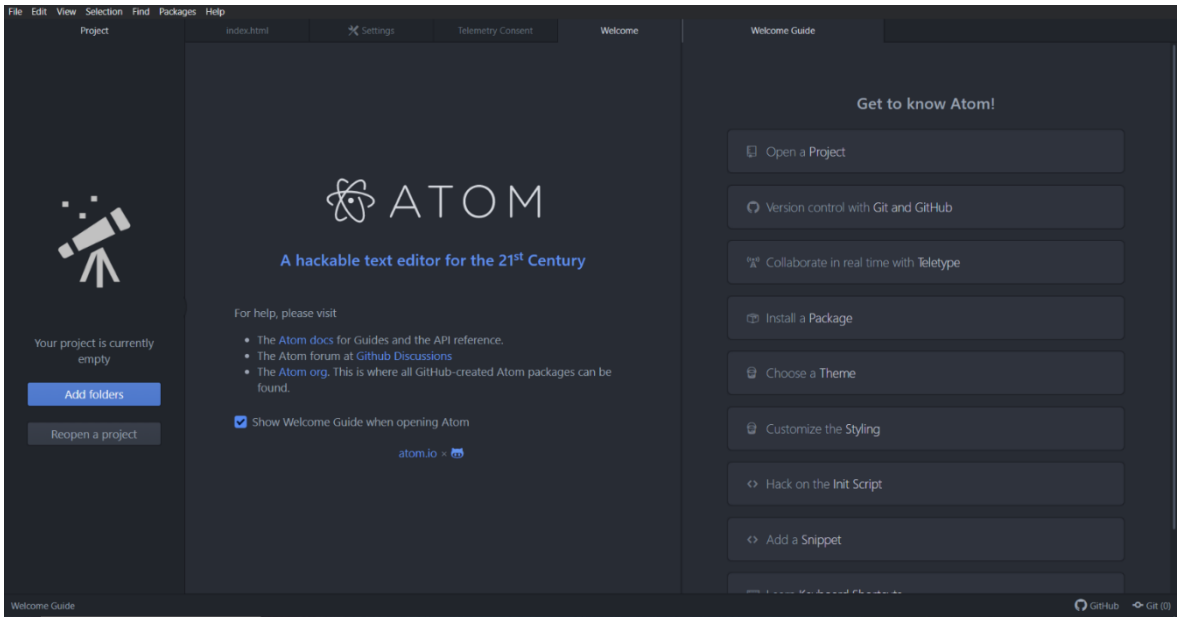


Рис. 1.6. Atom

WebStorm – це інтегроване середовище розробки компанії JetBrains для мов JavaScript, HTML, CSS розроблена на основі платформи IntelliJ IDEA. WebStorm постачається зі попередньо встановленими плагінами для JavaScript. Ця середа розробки підтримує мови JavaScript, CoffeeScript, TypeScript та Dart.

WebStorm має функціонал автодоповнення, аналізу коду, навігацію по коду, рефакторинг, дебагування, підтримка інструментів для написання тестів та інтеграція с системами управління версіями. Також є можливість розширення існуючого функціоналу за допомогою встановлення плагінів.

Робочий простір WebStorm відображений на рисунку 1.7.

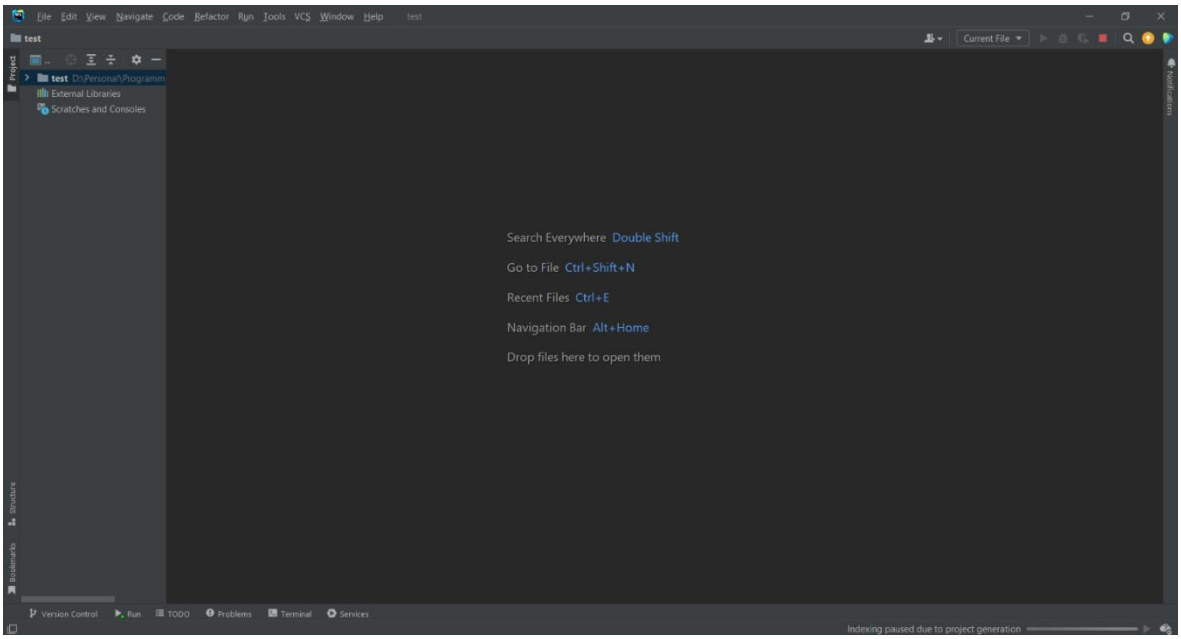


Рис. 1.7 Робочий простір WebStorm

Отже, для розробки додатку було обрано WebStorm, оскільки даний застосунок є повноцінним середовищем розробки зі зрозумілим інтерфейсом, швидкою обробкою коду та його форматування та відладки, попередньо встановленими необхідними бібліотеками та системою керування версіями для повноцінного та зручного процесу розробки.

2 ВИЗНАЧЕННЯ ВИМОГ ТА ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА АРХІТЕКТУРИ СИСТЕМИ

2.1 Визначення та моделювання вимог до web-застосунку

Задача дипломної роботи полягає у розробці веб-застосунку для планування романів, з можливістю структуризації інформації відповідно до певної книги та з вбудованими шаблонами для форм створення книги, персонажа, сцени.

В результаті проведеного аналізу існуючих аналогів сформовано функціональні та нефункціональні вимоги до програмного забезпечення.

Нефункціональними вимогами до web-застосунку визначено:

- мінімалістичний дизайн інтерфейсу з використанням стандартних іконок для взаємодії, для забезпечення використання застосунку користувачем, без необхідності навчання;
- сумісність з різними пристроями: web-застосунок має бути доступним для використання на різних пристроях, таких як комп'ютери з операційною системою Windows, MacOS, Linux, планшети та смартфони з операційною системою Android та IOS;
- доступність через усі сучасні браузери такі як: Chrome від версії 107.0.5304.63, Opera від 100.0.4815.20/ версії, Firefox починаючи з версії 107.0, Safari версії 5.1.10 тощо;
- середня швидкість обробки запитів у web-застосунку 20 ms;
- розробити наступні категорії до кожної книги: Description – опис книги, Characters list – список персонажів книги, Synopsis – короткий опис сюжету книги, Scene list – список сцен, Note – список нотатків;

– реалізувати створення та редагування книги, персонажа та сцени через форми, що мають структуру шаблону для створення відповідно кожного з об’єктів.

Функціональні вимоги :

– авторизація користувача за допомогою Google аккаунту;
– створення, редагування, видалення книги;
– створення, редагування, видалення персонажів сцен;
– створення, редагування, видалення сцен;
– створення, редагування, видалення нотатки;
– створення, редагування Description;
– створення, редагування Synopsis.

Отже, для початку роботи з web-застосунком користувач має зареєструватись за допомогою Google акаунту. Процес реєстрації та входу виконується лише один раз. У подальшому необхідності повторного входу не виникає, оскільки програма автоматично заходить до облікового запису користувача за його акаунтом, що був вказаний.

Авторизований користувач, запускаючи web-застосунок, завантажує головну сторінку, де він може створювати об’єкти книг за допомогою flow-кнопки з іконкою «+». Для створення книги, необхідно заповнити форму з шаблоном: поле для назви книги “Book name”, назву серії книг “Book series name”. Окрім цього, є можливість завантажити зображення в якості обкладинки книги.

Після створення книги користувач бачить картку створеної книги на полі основної сторінки книги. При натисканні на картку книги користувач переходить на її сторінку. На цій сторінці відображається раніше введена інформація про книгу у верхній частині сторінки. У нижній частині наявне робоче вікно з категоріями, в якості меню, з готовими шаблонами.

Категорія Description являє собою поле для вводу, куди користувач вводить з клавіатури текст короткого опису сюжету книги. Текст можна редагувати. Для збереження введеної інформації необхідно натиснути на кнопку «SAVE»;

Категорія Character list - це список, що відображає створених персонажів в якості невеликих карток з зображенням та ім’ям персонажа. Для створення нового персонажа необхідно натиснути на кнопку "ADD". Після натискання відкривається вікно з формою з шаблоном для створення персонажа. В цій формі наявні наступні поля для заповнення: Name, Role, Appearance description, Temper, Biography, Image. Для редагування інформації існуючого персонажа, необхідно натиснути на картку персонажа у списку та змінити інформацію у формі в модульному вікні. Для видалення існуючого персонажа, необхідно натиснути на іконку смітника на сторінці категорії на картці персонажа;

Категорія Synopsis являє собою поле для вводу для написання узагальненого опису всього сюжету книги. Текст можна редагувати. Для збереження введеної інформації необхідно натиснути кнопку «SAVE»;

Категорія Scene list - це список, що відображає описані сцени в якості невеликих карток. При натисканні на текст "ADD" відкривається вікно з формою з шаблоном для опису сцени. Ця форма має наступні поля для заповнення: Scene name для назви сцени та Scene description для введення опису сцени. Для редагування сцени, необхідно натиснути на картку зменшеного відображення сцени на сторінці категорії та змінити інформацію у формі. Видалення відбувається після натискання на іконку смітника на картці;

Категорія Note – список нотатків, що відображаються в якості карток під текстовим полем на сторінці категорії. Для створення нової нотатки необхідно ввести текст у текстове поле на натиснути на кнопку «SAVE». Для редагування необхідно натиснути на іконку олівця на картці нотатки, тіло замітки з’явиться у текстовому полі де можна змінити текст. Видалення відбувається після натискання на іконку смітника на картці нотатки на сторінці категорії.

Використання програми та взаємодія користувача з ним представлена Діаграма варіантів використання на рисунку 2.1.

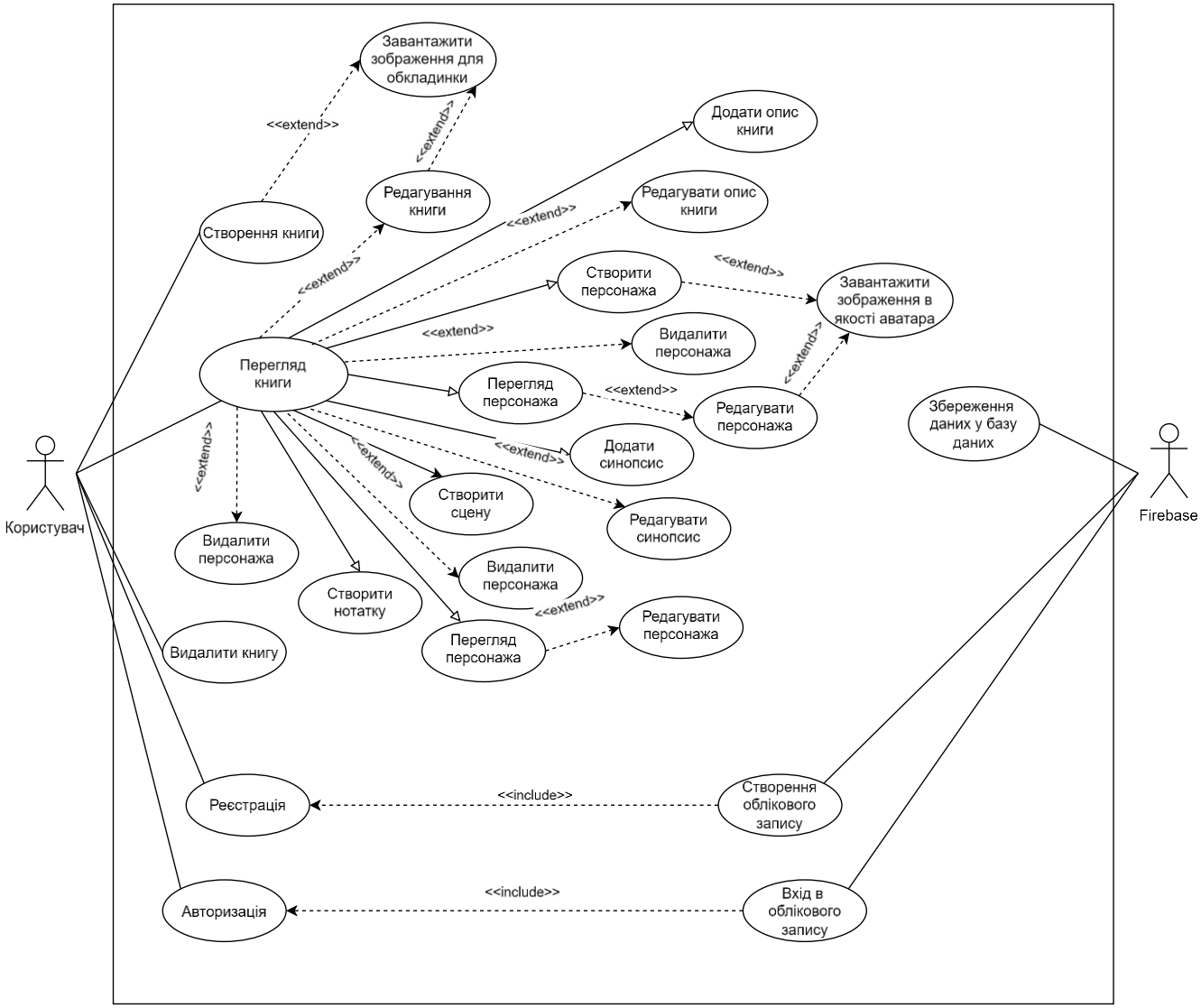


Рис. 2.1 Діаграма варіантів використання

Схема представлена на рисунку 2.2 відображає роботу web-застосунку.

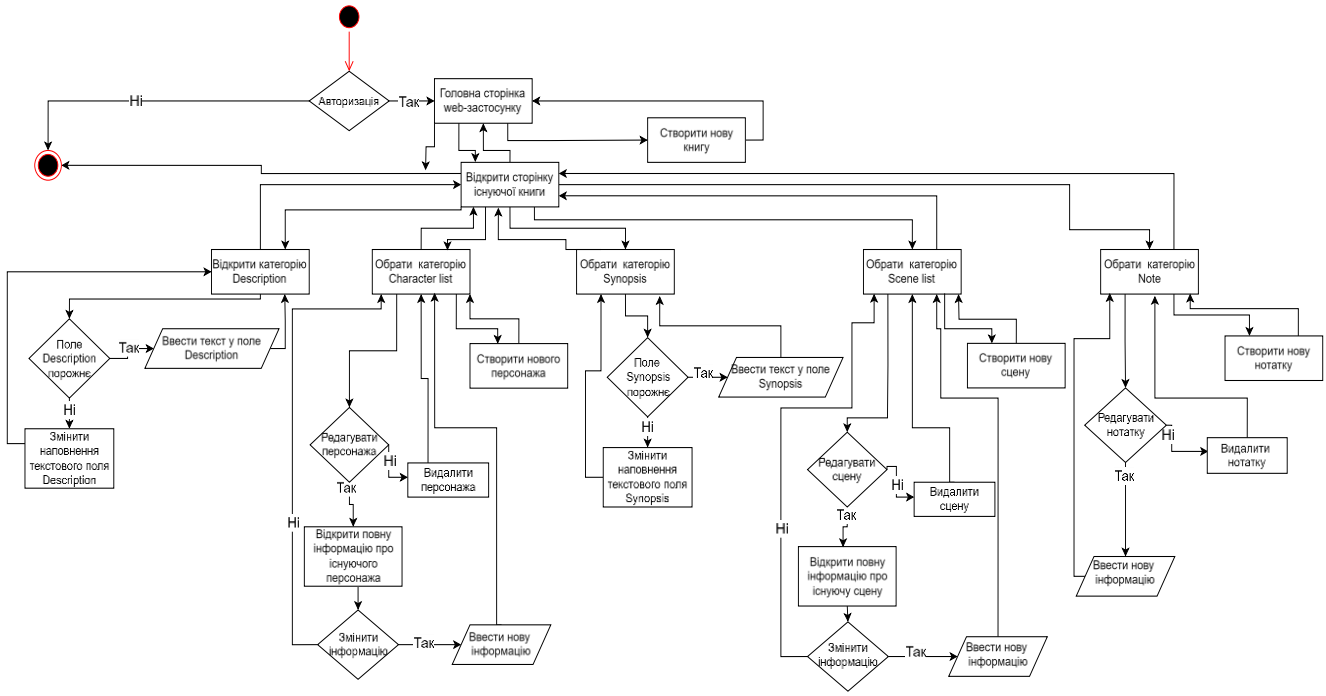


Рис. 2.2 Схема роботи web-застосунку

2.2 Проектування прототипу інтерфейсу користувача

Проектування інтерфейсу користувача проводилось у графічному редакторі Figma.

При розробці інтерфейсу необхідно зважати на наступні критерії:

- Зрозумілість – необхідно орієнтуватись на потенційних користувачів, використовувати зрозумілі назви та іконки для навігації;
- Простота – використання зрозумілих термінів, зменшення кількості кроків для виконання дій;
- Консистентність – дотримуватись одного стилю у всьому додатку;
- Адаптивність – для веб-додатку важливо, щоб інтерфейс був адаптований для різних розмірів екрану;

Окрім вище зазначеного, дизайн застосунку має бути приємним для сприйняття та не навантажувати очі користувача.

Під час планування прототипу визначено основні сторінки та елементи, що мають бути на них. По-перше, необхідно визначити вигляд головної сторінки. Вона має 2 стани. Перший стан представлений на рисунку 2.3 – порожня сторінка з назвою застосунку у шапці та кругла кнопка для створення книги у правому нижньому куті. Другий стан – сторінка зі створеними об’єктами книг рисунок 2.4.

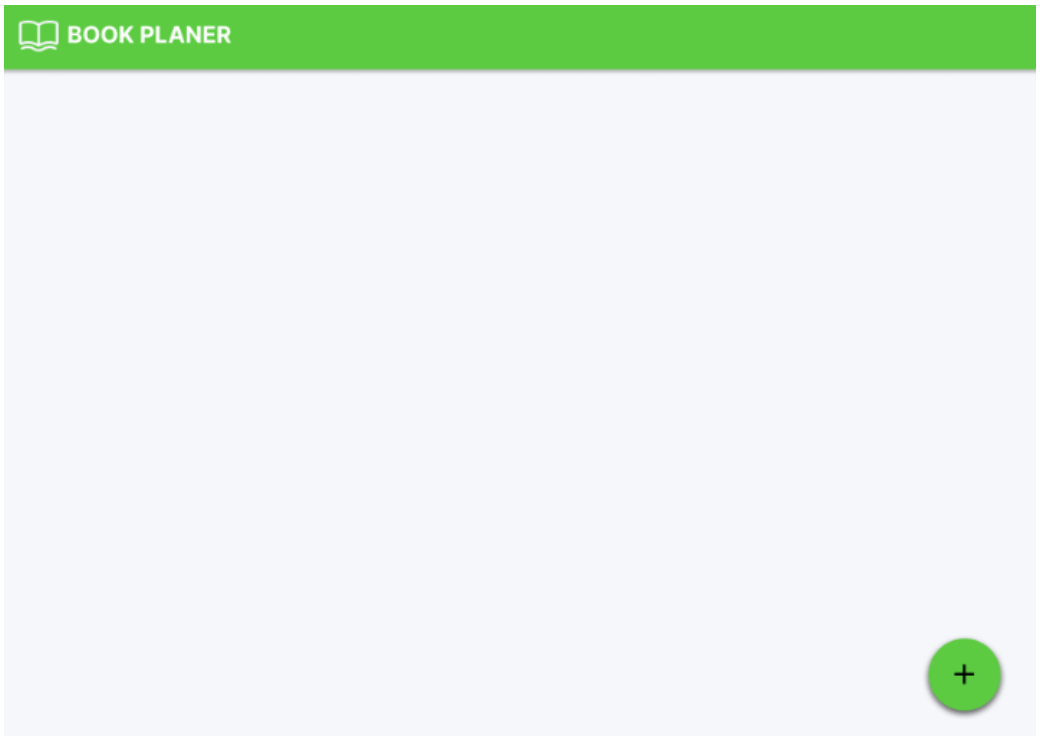


Рис. 2.3 Порожня головна сторінка

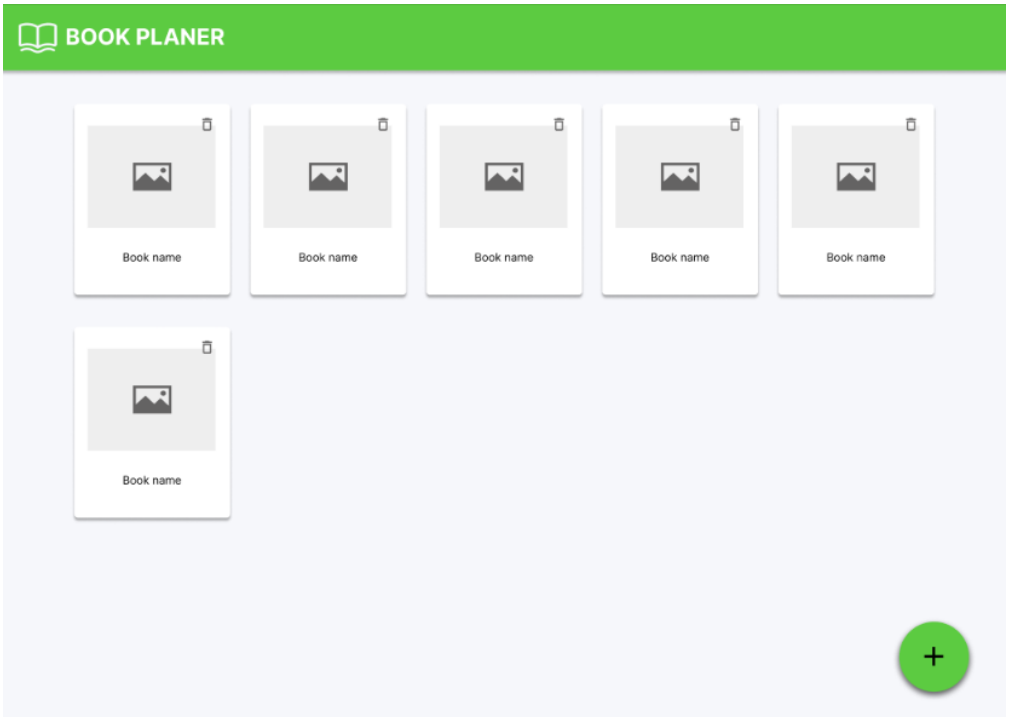


Рис. 2.4 Головна сторінка зі створеними книгами

Також, необхідно створити вікно створення книги. Воно має з’являтись зверху основної сторінки, затемнюючи її. Дане вікно міститиме поля для вводу назви книги з підписом “Book name” та назви серії книг з підписом “Book series name”. При натисканні на поле для написання назва поля переходить на лівий верхній кут поля. Також, у цьому вікні є можливість завантажити зображення в якості обкладинки. Нижньому правому кутку є дві кнопки підтвердження “SUBMIT” або відхилення “CLOSE” збереження введених даних. Прототип відображення представлений на рисунку 2.5.

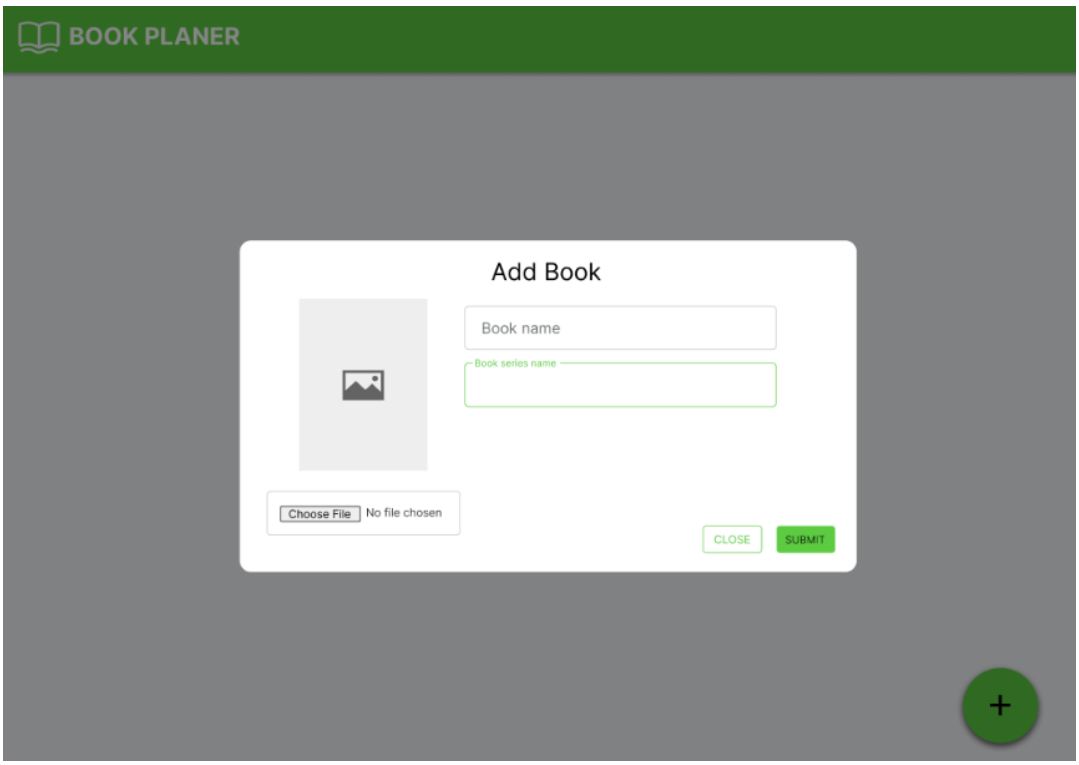


Рис. 2.5 Форма створення книги

Після створення книги відкривається її сторінка. Вона візуально поділяється на дві частини. У верхній частині відображається інформація про книгу, яка була введена у вікні створення книги, а також кнопка “EDIT” у правій частині екрану, для відкриття модального вікна редагування книги. У вікні редагування можна змінити попередньо введену інформацію, а також змінити обкладинку книги. За допомогою кнопок “SUBMIT” та “CLOSE” відбувається підтвердження на збереження нової інформації або повернення до попередньої. Форма представлена на рисунку 2.6.

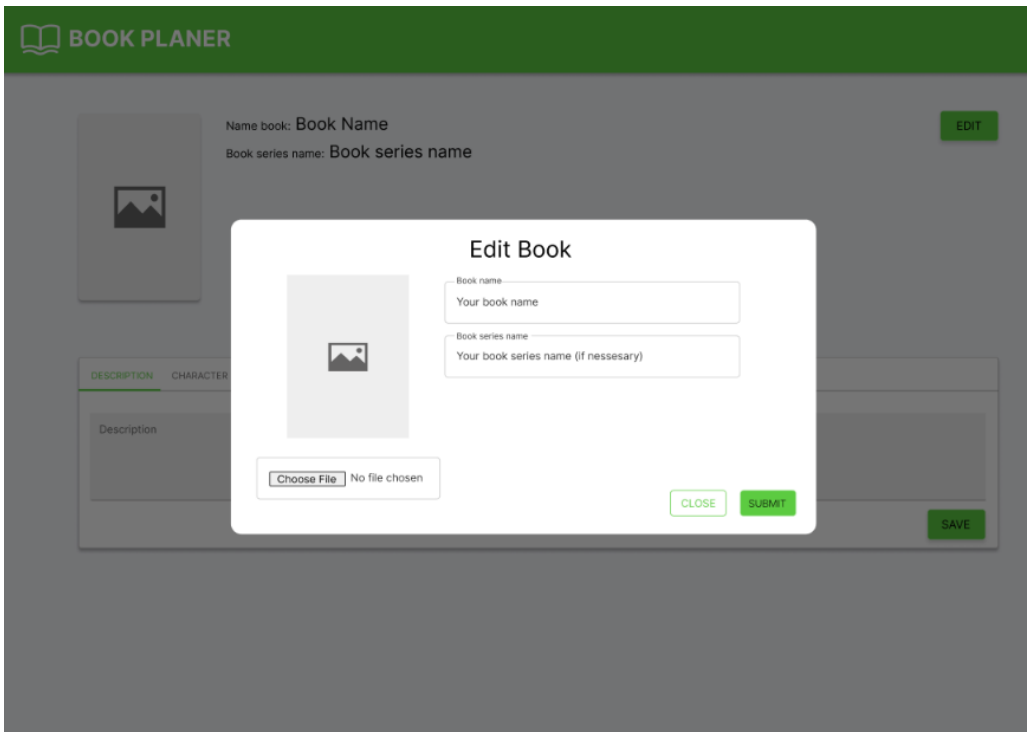


Рис. 2.6 Вікно редагування книги

У нижній частині розташоване інтерактивне вікно, що складається з горизонтального меню у верхній частині та полем з відображенням інформації під ним. Першим із пунктів меню є “Description” для опису книги та її ідеї, який представлений у вигляді текстового поля для вводу. Для збереження введеної інформації є кнопка “SAVE”. Вигляд форми представлено на рисунок 2.7.

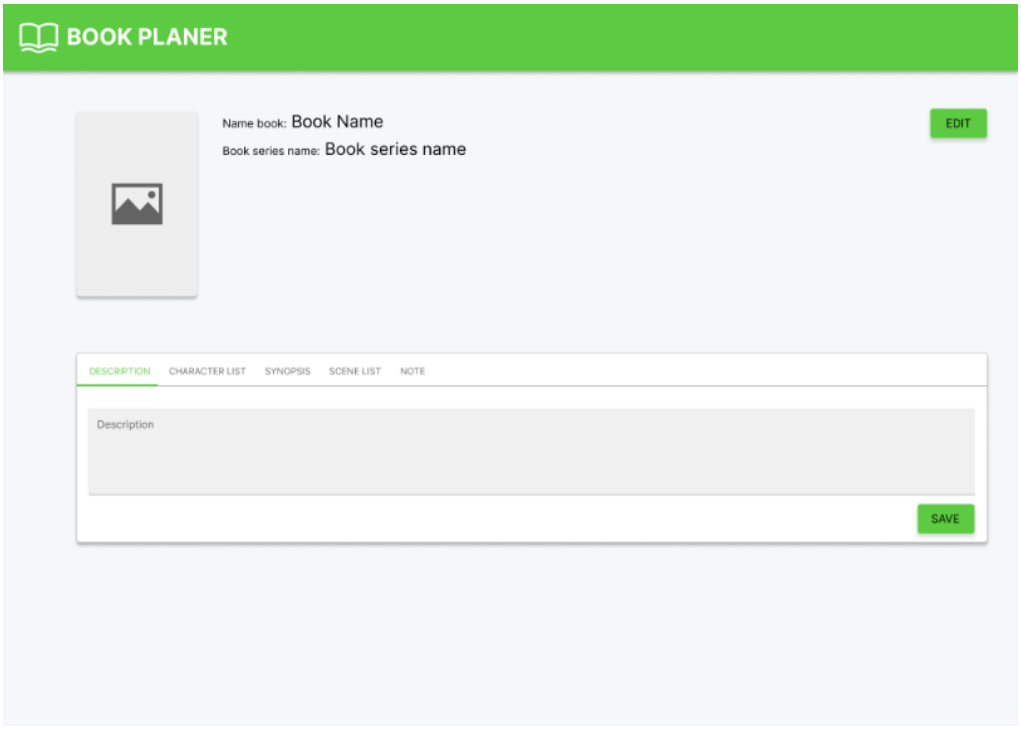


Рис. 2.7 Опис книги

Другий пункт меню являє собою перелік персонажів. У вікні відображення є надпис “ADD”, натиснувши на який відкривається вікно створення персонажа. Під надписом відображаються створені персонажі. Кожна картка персонажа показує завантажене зображення або стандартне, якщо не було завантажене власне. Під зображенням показане ім’я персонажа та іконка смітника для видалення. Відображення на сторінці представлене на рисунку 2.8.

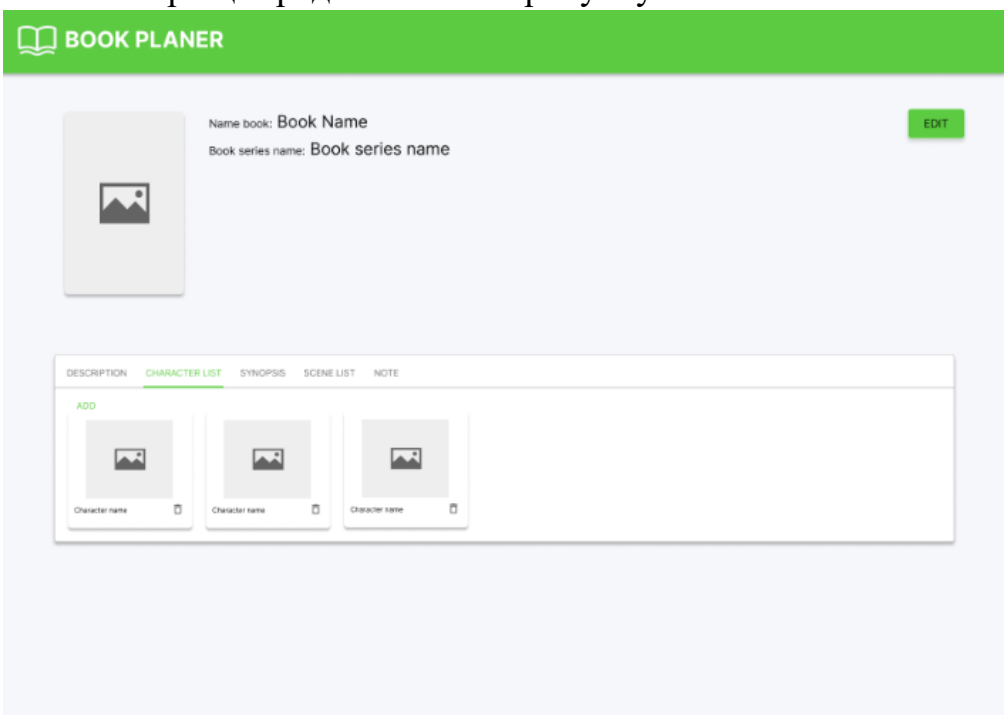


Рис. 2.8 Перелік персонажів

У вікні створення персонажа відображається форма. У ній є наступні поля для заповнення: поле для імені “Name”, вказання ролі персонажа “Role”, для описання зовнішності “Appearance description”, опис характеру персонажа “Temper” та “Biography” для написання біографії. Окрім цього, є можливість завантажити зображення, в якості аватару, за допомогою кнопки “Choose File”. Для керування збереженням інформації є кнопки “SUBMIT” та “CLOSE”. Прототип форми відображений на рисунку 2.9.

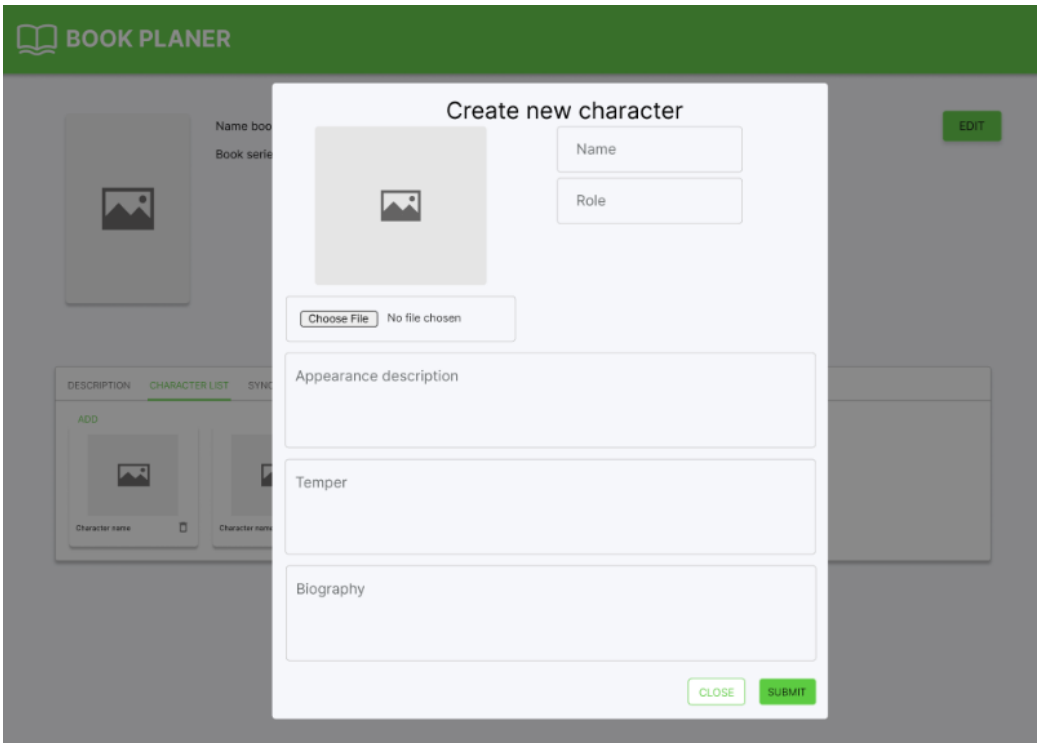


Рис. 2.9 Вікно створення персонажа

Для редагування інформації уже створеного персонажа, після натискання на його карточку, відкривається модальне вікно редагування. Для керування збереженням інформації є кнопки “SUBMIT” та “CLOSE”. Модальне вікно редагування відображене на рисунку 2.10.

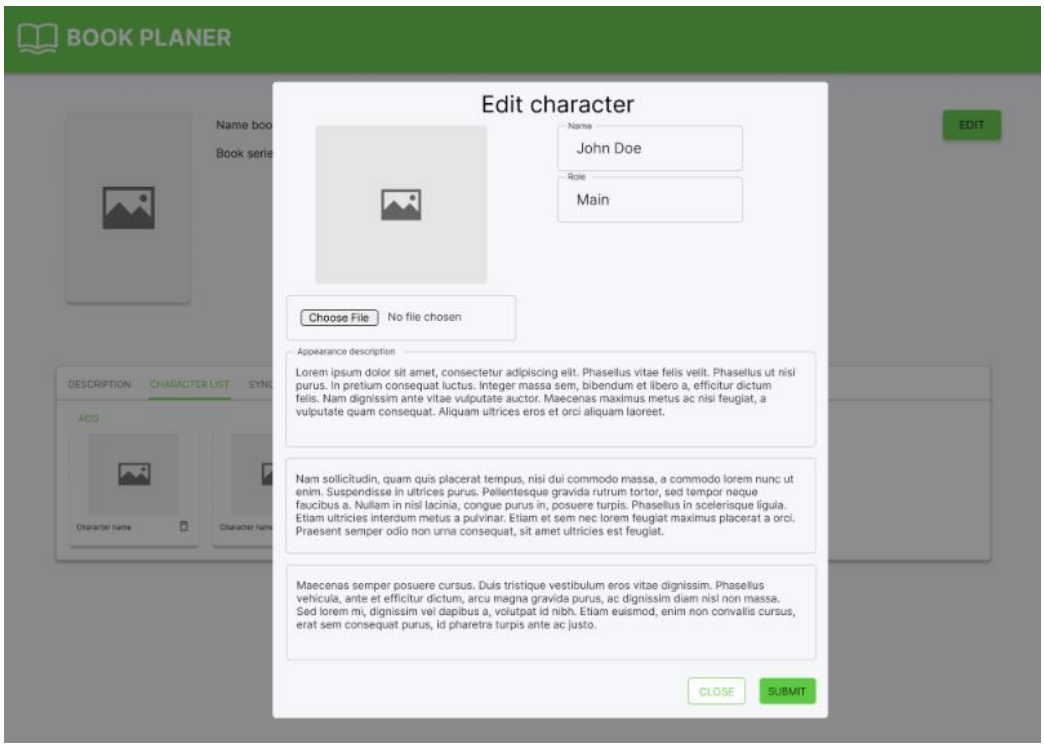


Рис. 2.10 Вікно редагування персонажа

Третім пунктом меню являється Синопсис – стислий переказ загального сюжету книги, який використовується при звертанні у видавництво або подання роботи на конкурс. У додатку представлений у вигляді текстового поля для вводу. Для збереження введеного тексту є кнопка “SAVE” у нижній правій частині вікна. Вигляд сторінки категорії представлено на рисунку 2.11.

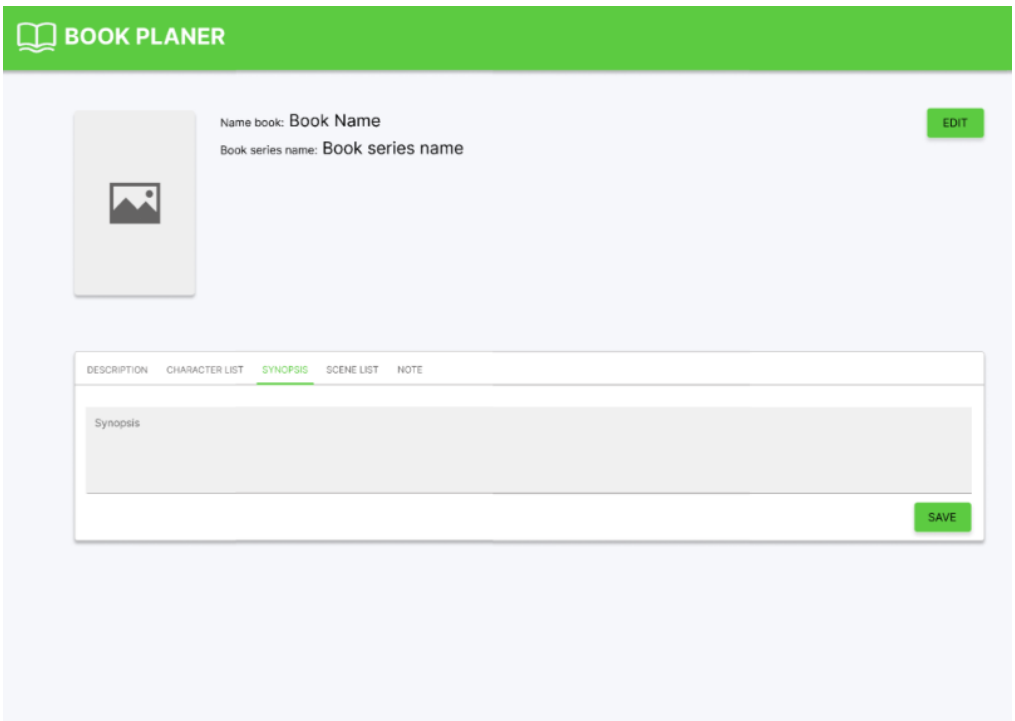


Рис.2.11 Синопсис

Наступний пункт – список сцен. На полі під меню є кнопка “ADD” під нею будуть відображатись створені сцени. Кожна картка показує назву сцени, у правій частині картки іконка смітника для видалення сцени представлені на рисунку 2.12.

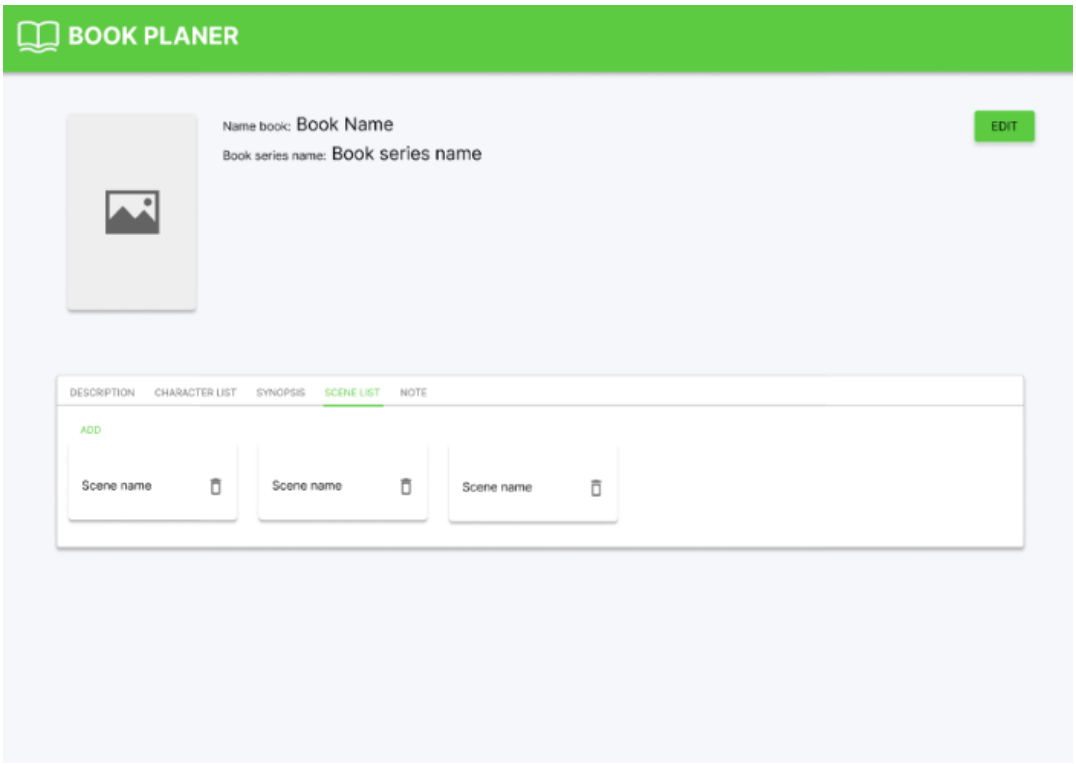


Рис. 2.12 Список сцен

При натисканні на “ADD” відкривається вікно з формою створення сцени. Форма має наступні поля для заповнення: “Name” для введення назви сцени, а також “Scene description here…” для детального опису сцени. Для збереження є кнопка “Submit”. Щоб відмінити збереження введених даних – кнопка “Close”. Форма створення представлена на рисунок 2.13.

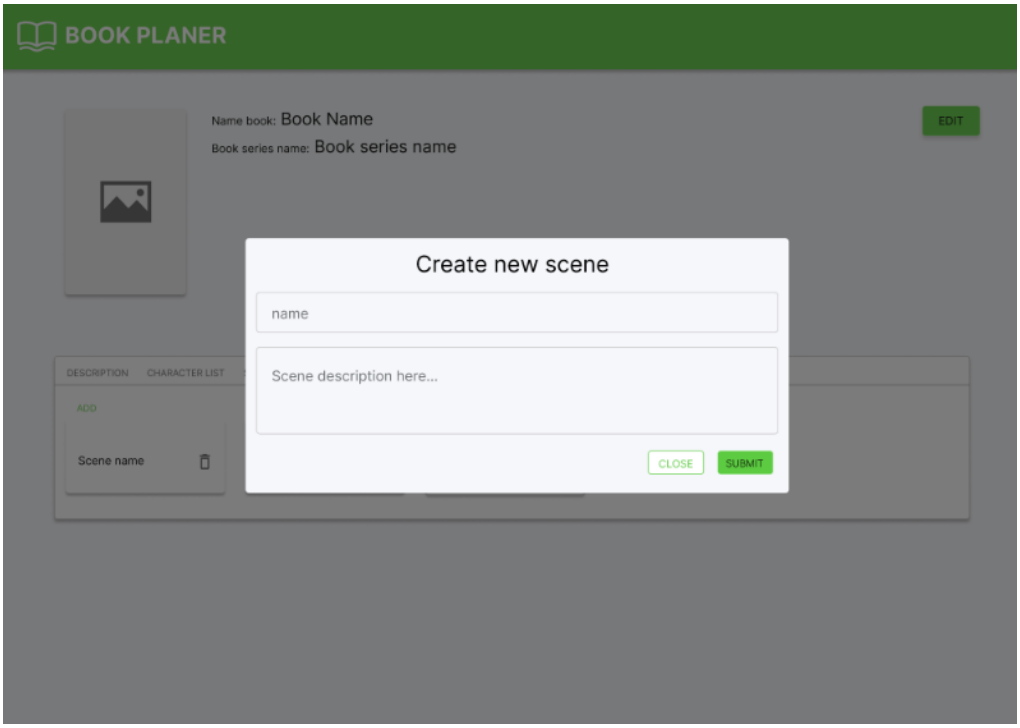


Рис. 2.13 Вікно створення сцени

Для зміни сцени, при натисканні на необхідну картку, відкривається модальне вікно редагування. У цьому вікні можна змінити назву та опис сцени. Збереження нової інформації - “SUBMIT”, для відміни внесених змін використовується кнопка “CLOSE”. Рисунок 2.14.

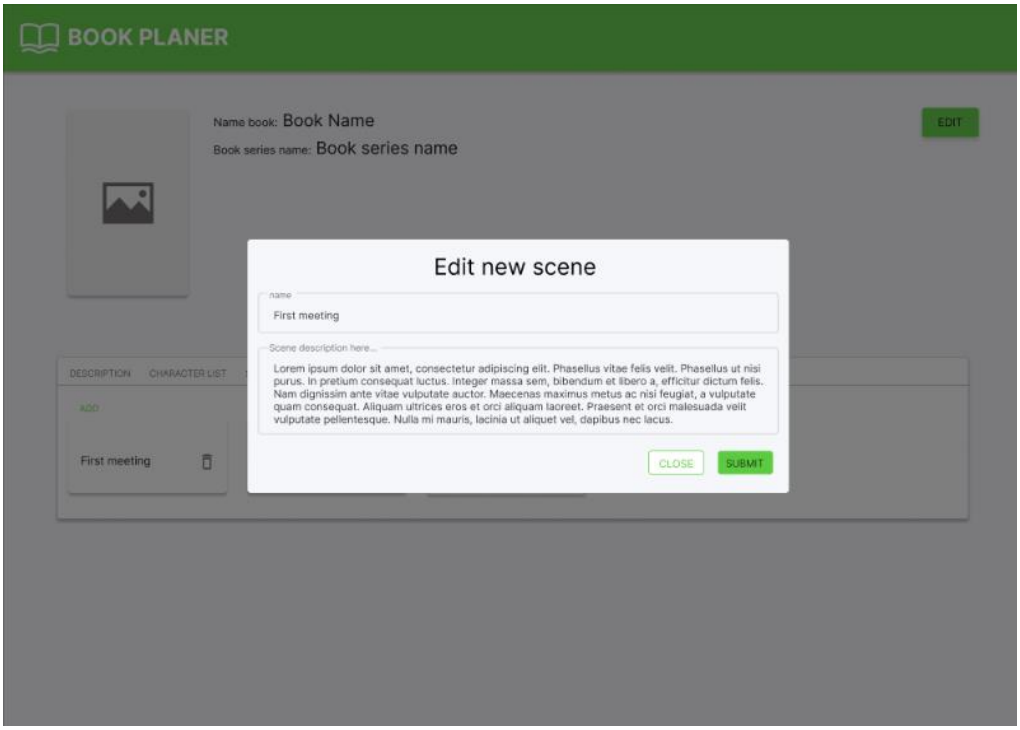


Рис. 2.14 Вікно редагування сцени

Останнім пунктом є “Note” для створення нотатків. Данна сторінка в меню представлена текстовим полем для вводу тіла нотатка. Під ним справа знаходиться кнопка для збереження “SAVE”. Створенні нотатки відображаються у вигляді карток від текстовим полем. Кожна картка відображає тіло нотатки, під тілом зліва показується дата та час її створення, зліва знаходяться іконки редагування та видалення. Порожня сторінка відображена на рисунки 2.15, на рисунку 2.16 показана сторінка зі створеним нотатком.

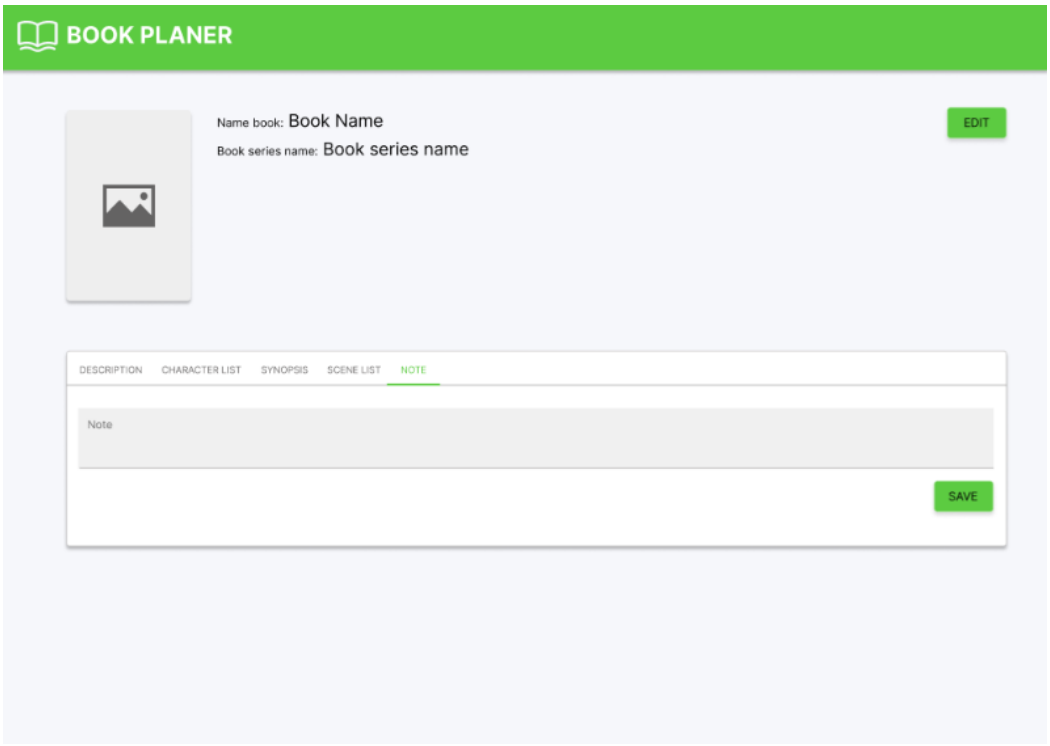


Рис. 2.15 Порожня сторінка нотатків

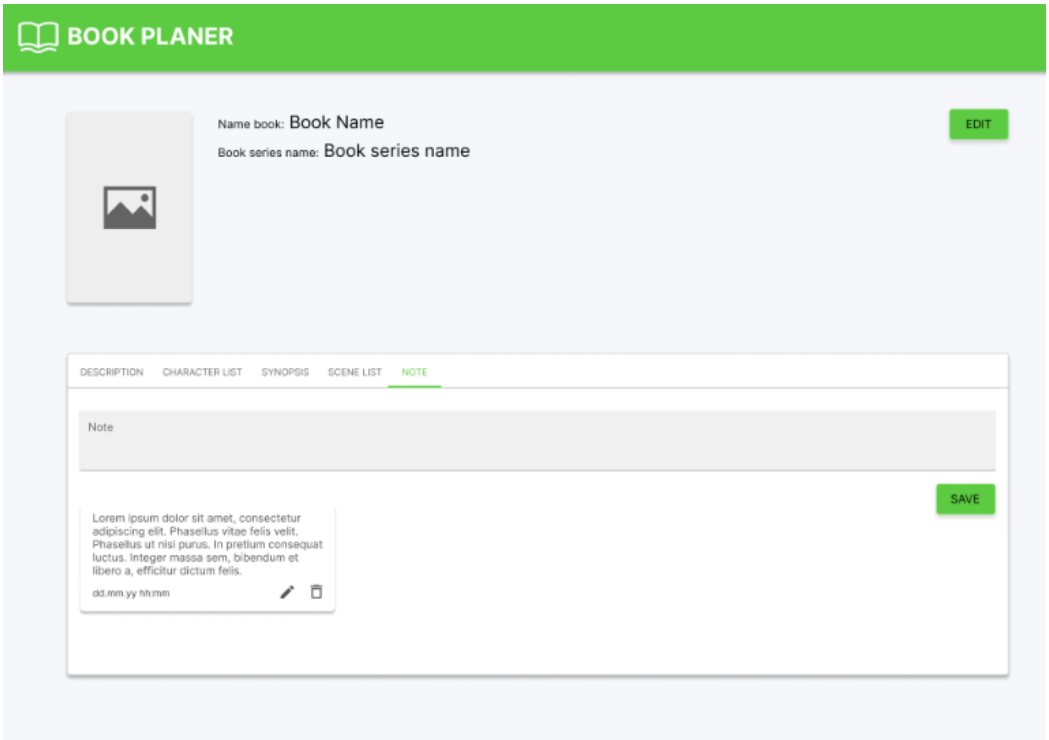


Рис. 2.16 Сторінка нотатків зі створеним об'єктом

Редагування нотатків відбувається після натискання на відповідну іконку олівця. Текст відображається у текстовому полі для введення. Для збереження використовується кнопка “SAVE”. Редагування представлено на рисуноку 2.17.

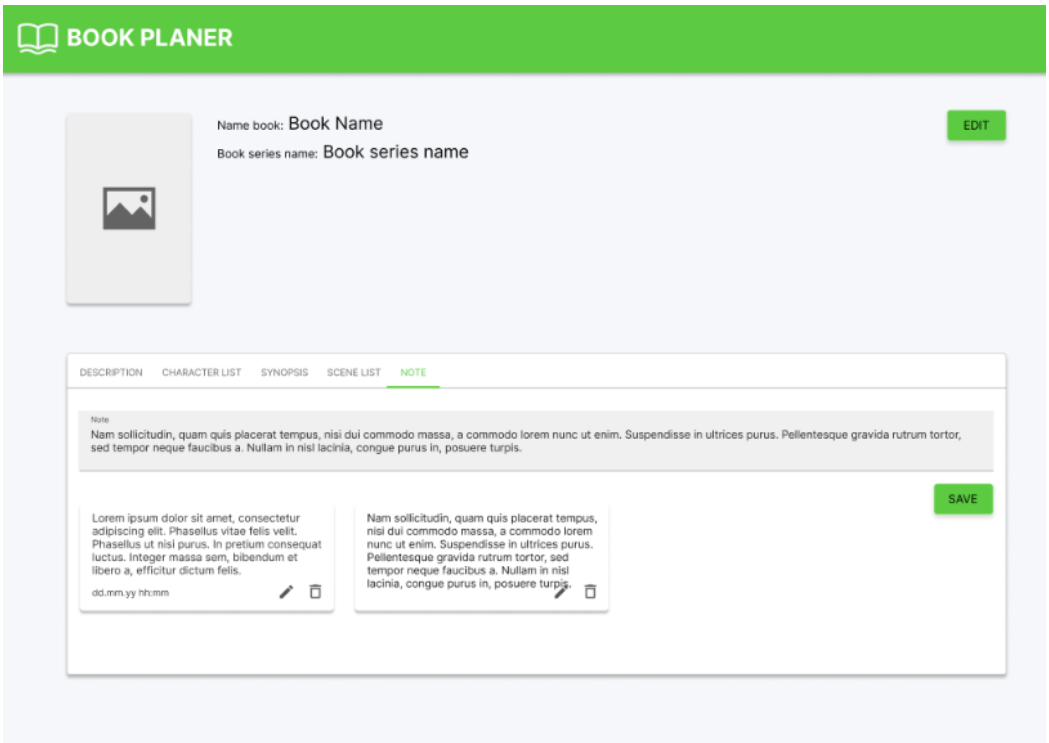


Рис. 2.17 Редагування нотатку

2.3 Проектування структури web-застосунку

2.3.1 Структура клієнтської частини web-застосунку

Клієнтська частина реалізована з використання React, що надає можливість створювати компоненти, що можуть бути використані повторно. Бібліотека React надає доступ до безлічі бібліотек для конструювання інтерфейсів. Для розробки інтерфейсу додатку «Book planner» використана бібліотека Material UI – надає набір різних утиліт для створення Material Design System інтерфейсів. Дана бібліотека має ряд переваг, таких як: можливість створення адаптивних інтерфейсів під різні роздільні здатності екрану; готові компоненти з прописаним дизайном. Також, MUI підтримує усі сучасні та популярні браузери.

Дана частина представлена п’ятьма класами.

Клас Books у якому представлено масив books, що складається з об’єктів класу BookItem.

Клас BookItem, що зберігає інформацію про створену книгу у string полях img, name, series_name, userId – унікальне значення авторизованого користувача, а також масиви BookCharter, BookNotes, BookSceneList.

Клас BookCharter несе у собі інформацію про персонажа книги. Інформація записана у полях appearance – тип string, зберігає інформацію про зовнішній вигляд героя; biography – тип string, несе відомість про біографію персонажу; img – тип поля string, зберігає посилання на зображення, що завантажене у якості аватару персонажу; name – тип string, зберігає ім’я персонажу; role – поле типу string, що несе інформацію про роль персонажа у книзі; temper – тип string, що зберігає інформацію про характер персонажу.

Клас BookNotes зберігає інформацію про нотатку. Дані зберігаються у полях типу string: text – несе у собі тіло нотатки; created – зберігає дату та час створення нотатки.

Клас BookSceneList створює об’єкт сцени книги. Інформація зберігається у полях description – тип string, зберігає опис сцени та name – поле типу string, що запам’ятовує назву сцени.

Клієнтська частина представлена у вигляді діаграми класів на рисунку 2.18.

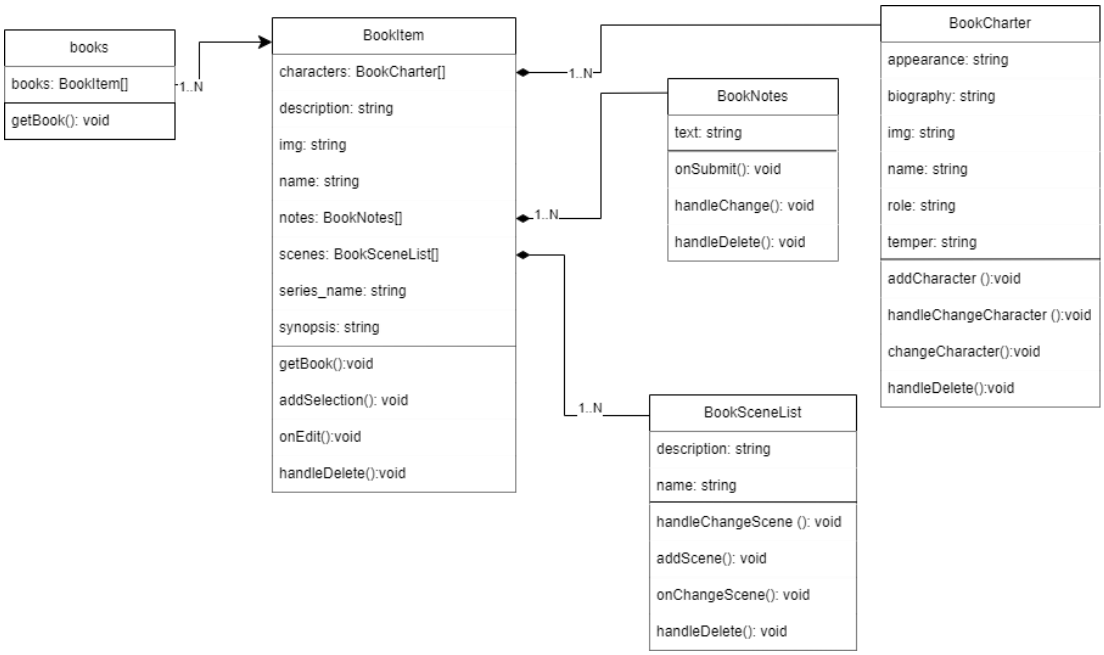


Рис. 2.18 Діаграма класів

2.3.2 Структура серверної частини web-застосунку

Серверна частина web-застосунку реалізована завдяки Firebase. Серверна інфраструктура, що реалізована завдяки Firebase, надає можливість розгортання додатку через Firebase Hosting, який дозволяє швидко та легко розгорнути web-застосунок через командній рядок. Окрім цього, hosting надає можливість оновлювати інформацію у базі даних в режимі реального часу, а використання

глобальної мережі CDN (Content Delivery Network) виконує швидку доставку інформації від сервера до клієнта.

Також використовується Firebase Authentication. Основними перевагами даного сервісу є можливість простого та безпечного створення користувачами облікових записів, входження до облікових записів та використовувати різні методи для аутентифікації такі як Facebook, Google, GitHub тощо. Для web-застосунку було обрано використати реєстрацію за допомогою Google. Також Firebase Authentication надає унікальне ім’я кожного користувача, що дає змогу зберігати асоційовані з ними дані в базі даних та визначати користувача, та захищає дані від атак перехоплення сесій тощо. Окрім цього, його зручно використовувати з іншими сервісами Firebase, які також використовуються у web-застосунку.

База даних представлена Firebase Firestore, що є NoSQL базою даних. Усі дані в базі організовані у вигляді колекцій та документів. Усі документи представлені об’єктами, що зберігають інформацію у форматі ключ-значення. В свою чергу колекції є групами таких документів. Така структура організовує дані відповідно до логічних зв’язків та дозволяє швидко знаходити та діставатись до них. Окрім цього, Firestore оновлюється у режимі реального часу та не потребують додаткових дій для оновлення інформації. Дана база даних забезпечує вбудовану безпеку з механізмом контролю доступу до даних, правила доступу до даних можуть бути налаштовані відповідно до вимог додатку.

Структура бази даних представлена на рисунку 2.19.

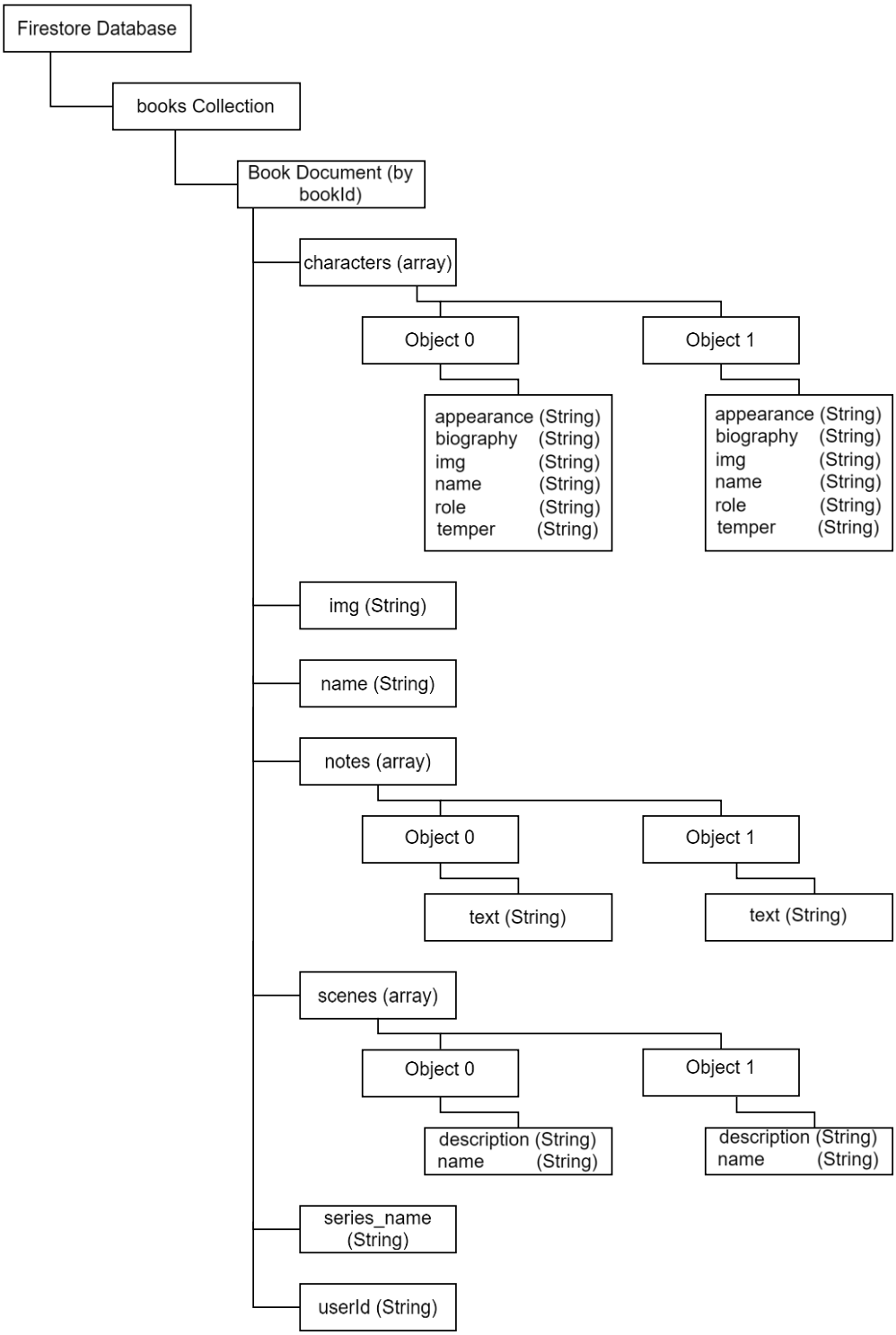


Рис. 2.19 Діаграма бази даних

3 РОЗРОБКА WEB-ЗАСТОСУНКУ

3.1 Опис інструментів реалізації

Для розробки web-застосунку було обрано наступні мови: JavaScript, HTML, CSS.

JavaScript – одна в найпоширеніших мов програмування web-розробки. Це мова скриптів, що використовується для створення інтерактивності та взаємодії зі сторінкою, що робить її ідеальним вибором для програмування web-сторінок на стороні клієнта.

JavaScript має вбудовані перевірки заповнення форм, змушує браузер запам’ятовувати інформацію про користувача до наступного використання, створює інтерактивні віджети для інтерфейсу. Дана мова дозволяє маніпулювати об’єктами на сторінці застосунку, додавати стилі до них.[3]

Має синтаксис схожий до С-подібних мов, що робить її достатньо зрозумілою для багатьох програмістів. JavaScript базується на принципах об’єктно-орієнтованого програмування (ООП), вона підтримує класи, об’єкти, наслідування, інкапсуляцію та інші концепції ООП. Дана мова підтримує динамічну типізацію змінних, а отже тип змінної визначається автоматично та може змінюватись під час роботи програми.

Також, JS здатна динамічно змінювати елементи HTML, властивості та стилі web-сторінки, обробляти події від користувача, як натискання кнопок, притягнення об’єктів.[4]

Окрім цього JavaScript підтримує використання сторонніх бібліотек та фреймворків, що дозволяє розширювати його базові можливості.

HTML (Hypertext Markup Language) – є основною мовою розмітки для створення та відображання веб-сторінки. Використовується створення структури документа, в якому буде розміщуватись вміст сторінки. Всі елементи HTML представляють собою теги, що оточують текст та інші елементи та надають їм певні характеристики чи функціонал.

Основними поняттями HTML є теги, атрибути, елементи, структура документа.[5]

Теги – спеціальні елементи розмітки, що визначають як документ HTML буде інтерпретований та показаний на сторінці користувача. Зазвичай теги виглядають як пара відкриваючий <назва тегу>, вказує на початок елемента та може містити атрибути, та закриваючий </назва тегу>, вказує на кінець елемента. Деякі теги можуть бути однозначними та не потребують закриваючого тегу, як наприклад <img>.

Атрибути – це додаткова інформація до тегу, яка може внести деякі зміни до їх стилю, характеристик або вказівок для браузеру, як необхідно взаємодіяти з цим елементом.

Основними властивостями атрибуту є назва – ім’я, яке ідентифікує кожен атрибут, значення – конкретне значення, що встановлює атрибут. Значення розділяється від назви знаком рівності ‘=’ та обертається у подвійні або одинарні лапки ‘ href="https://www.example.com" ’ .

Структура документу складається з заголовку (<head>) та тіла (<body>). У заголовку міститься інформація про документ, заголовок сторінки, метатеги, посилання на зовнішні ресурси тощо. У тілі відображується весь вміст сторінки.[6]

CSS (Cascading Style Sheets) – мова стилізації для визначення вигляду та форматування елементів на веб-сторінці. Вона дозволяє відокремити стиль веб-документа від структури, що дозволяє легко змінити зовнішній вигляд сайту без спливу на структуру контенту.[7]

Основними можливостями CSS є:

- вибір селекторів – селектори визначають, на які саме елементи сторінки будуть застосовані стилі.
- властивості та значення використовуються для задання конкретних характеристик елементів. Наприклад ‘color’ визначає колір тексту.
- каскадність – якщо стилі конфліктують, їх властивості визначаються відповідно до правилу каскаду – стилі можуть бути успадковані, перевизначені або складені в залежності від пріоритету.
- блокова модель допомагає визначити кожен елемент на сторінці своїми внутрішніми та зовнішніми розмірами. Включає у себе властивості як: ‘margin’, ‘padding’, ‘border’ тощо.
- позиціонування дозволяє контролювати розташування елементів на сторінці за допомогою методів позиціонування ‘position’, ‘float’, ‘display’.

React – бібліотека з відкритим кодом для веб-розробки, яка стала популярною у спільності розробників мовою програмування JavaScript. Вона була розроблена компанією, що підтримує Facebook та Instagram та використовується для створення ефективних динамічних інтерфейсів.

React розділяє інтерфейс на невеликі незалежні компоненти. Компонент – це невеликий блок, що містить HTML-код, логіку та стилі, що дозволяє створювати чистий код та використовувати його повторно.

Для ефективного оновлення сторінок, використовується віртуальний DOM. React створює віртуальну копію DOM та порівнює її з попередньою версією. Після порівняння оновлюються лише елементи, що були змінені. Окрім цього, оновлення відбуваються автоматично при зміні властивостей або стану за допомогою реактивного підходу до розробки.[8]

Розширення мови JavaScript JSX дозволяє включити HTML-подібний код безпосередньо до JS для пояснення React, як має виглядати інтерфейс.

React дозволяє розробляти односторінкові web-застосунки, на які завантажується лише необхідне наповнення та не перезавантажувати сторінку при переходах.

Для створення динамічних та інтерактивних інтерфейсів React використовує дві фундаментальні концепції – стан (state) та властивості (props).

Стан (state) – об'єкт, який використовується для керування та зберігання даних, що можуть змінитися в ході життєвого циклу компонента. Він використовується для ініціалізації стану, зміни стану, замиканні при асинхронності за допомогою 'setState' тощо.

Властивості (props) використовуються для передачі даних в компоненти та взаємодії між батьківськими та дочірніми компонентами. Після передачі, props залишаються незмінними в середині компонента. Їх можна використовувати у функціональних компонентах для отримання значень властивостей та визначення дефолтних значень, якщо значення не буде передано. Також, властивості надаються можливість налаштування поведінки та вигляду компонентів для багаторазового використання.[9]

Окрім цього, React має ряд додаткових інструментів та бібліотек, що можуть розшири його функціонал та зробити більш гнучким у використанні.

Отже, React дозволяє зробити процес розробки більш доступним, гнучким, швидким. Його підтримка величезним ком'юніті впливає на постійний розвиток та отримання нових бібліотек, що ще більше покращують процес розробки, якість коду та заощаджують час розробника. Віртуальний DOM покращує взаємодію зі сторінкою сайту та зменшує необхідність її постійного завантаження та перезавантаження для оновлення даних.

3.2 Опис реалізації аутентифікації

Для початку роботи з web-застосунком користувач має зареєструватись. Автентифікація відбувається за допомогою Google акаунту через вікно вибору акаунту при першому відкритті web-застосунку, оскільки вся інформація, зберігатиметься у Firebase Firestore під унікальним номером користувача. Перевірка користувача реалізована за допомогою Firebase Authentication. Для авторизації використовуються дві функції з пакету “firebase/auth”, а саме sighInWithPopup, яка викликає вікно вибору акаунту для аутентифікації та повертає об’єкт UserCredential в разі успішного виконання, та getAuth для отримання об’єкту аутентифікації для використання web-застосунку. Реалізація коду відображена на рисунку 3.1.

```
const AuthProvider = ({ children }) => {
  const auth = getAuth(firebaseApp);
  const [user, setUser] = useState(auth.currentUser);

  useEffect( effect: () => {
    auth.onAuthStateChanged((maybeUser) => {
      if (maybeUser !== null) {
        setUser(maybeUser);
      } else {
        signInWithPopup(auth, googleAuthProvider)
          .then((credentials) => setUser(credentials))
          .catch((e) => console.log(e));
      }
    });
  }, deps: [auth]);

  return user === null ? <Loader /> : children;
};

export default AuthProvider;
```

Рис.3.1 Реалізація аутентифікації користувача

3.3 Опис реалізації головної сторінки та функцій створення об’єкту книги

Після отримання доступу, користувачу відкривається головна сторінка web-застосунку. На ній відображається toolbar з назвою web-застосунку та кнопка створення нової книги в правому нижньому куті екрану. Створення та редагування нової книги реалізовано як компонент модального вікна у файлі AddModalBook. Дане модальне вікно представлене формою управління, що відбувається за допомогою методу useForm з бібліотеки “react-hook-form”. Метод getAuth отримує об’єкт аутентифікації та ініціалізує його для поточного з’єднання з Firebase. При завантажені нового файла, в якості обкладинки, викликається метод handleAddFile, який, у свою чергу, викликає функцію uploadFile, що завантажує файл на сервер. Для видалення зображення викликається функція deleteFile. Для збереження створеної книги, після натискання користувачем на кнопку SUBMIT, викликається метод onSubmit, який надсилає форму, створює новий документ до колекції book із введеними користувачем даними у форму. Після збереження викликається метод onClose, що відповідає за закриття форми та відновлення її до початкового стану.

Окрім створення нового об’єкта книги AddBookModal також відповідає за редагування вже створених об’єктів. Ключовими моментами в такому випадку є встановлений флаг isEdit у методі onSubmit, що оновлює існуючий документ у колекції.

Після створення об’єкту книги вона відображається на тілі головної сторінки, як представлено на рисунку 3.2. При натисканні на її картку відбувається перехід на сторінку для планування.

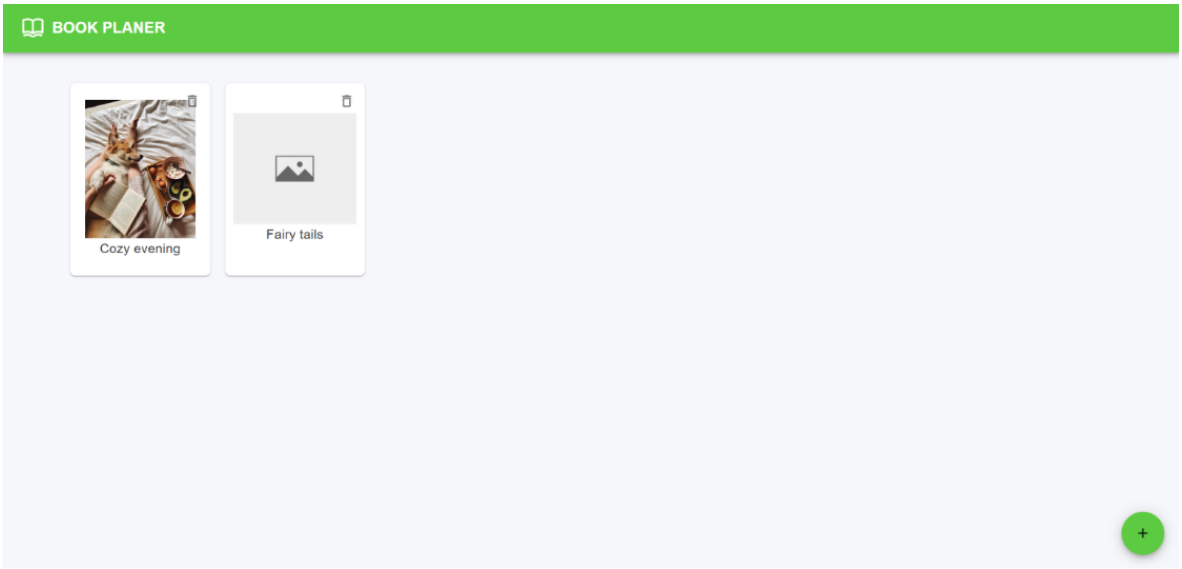


Рис. 3.2 Головна сторінка додатка з книгами

3.4 Опис реалізації сторінки книги та функцій для планування

Сторінка книги поділяється на дві частини. У верхній відображається інформація про книгу, яку вводили у форму при створені, а також кнопка EDIT, для редагування. Вікно редагування відображене на рисунку 3.3.

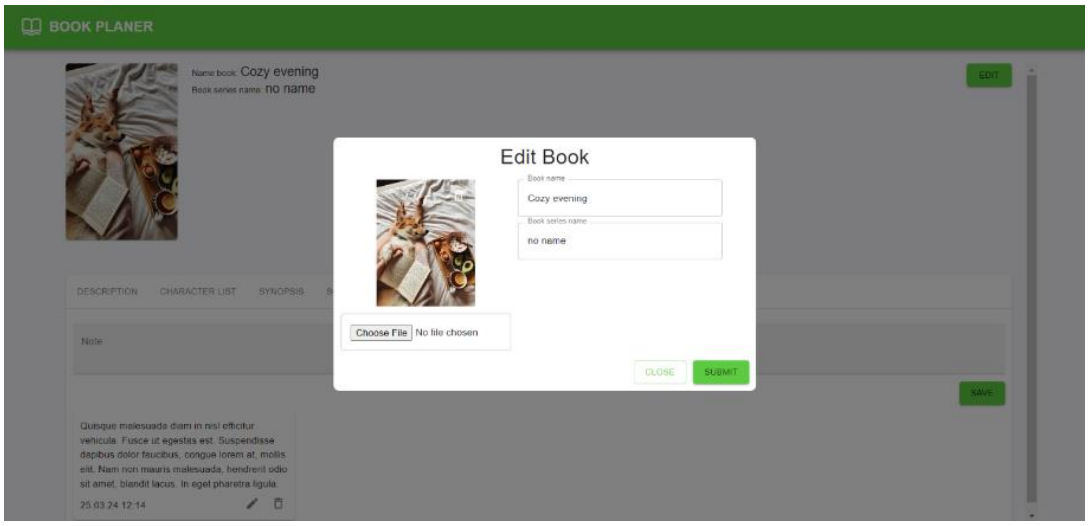


Рис. 3.3 Форма редагування книги

Нижня частина сторінки представлена робочим простором, що має вкладки з основними елементами для планування, такими як Description, Character List, Synopsis, Scene List, Notes. Для роботи з Description та Synopsis були створені відповідні компоненти BookDescription та BookSynopsis, що схожі у реалізації. Для управління формою використовується хук useForm з бібліотеки “react-hook-form”, який повертає об’єкт з даними та методами, що необхідні для роботи з формою. Для надсилання форми, після натискання користувачем на кнопку SAVE, викликається функція onSubmit, що викликає функцію addSelection з об’єктом data, що несе у собі текст введений у поле.

Для створення персонажів необхідно перейти на вкладку Character List та натиснути на кнопку ADD. Після цього відкривається модальне вікно з формою для створення персонажа реалізоване у компоненті BookCharterModal. Дана форма має наступні поля для заповнення: Name для имени персонажа, Role для визначення роли персонажа, Appearance Description – зовнішність персонажа, Temper для опису характеру персонажа та Biography для опису життя. Для збереження після натискання на кнопку SUBMIT викликається метод onSubmit, що перевіряє, якщо isEdit має значення false, то форма створює нового персонажа в колекції. У випадку, якщо користувач натискає на карту уже створеного персонажа, відкривається модальне вікно компоненту BookCharterModal та під час перевірки значення isEdit, яке буде дорівнювати true, буде відбуватись зміна раніше збереженої інформації на нову. Дана форма дозволяє додати зображення, що реалізується методом addImage. Окрім цього, було реалізовано метод handleClose, який відповідає за закриття модульного вікна та відновлення форми до початкового стану. Вигляд форми у додатку представлений на рисунку 3.4.

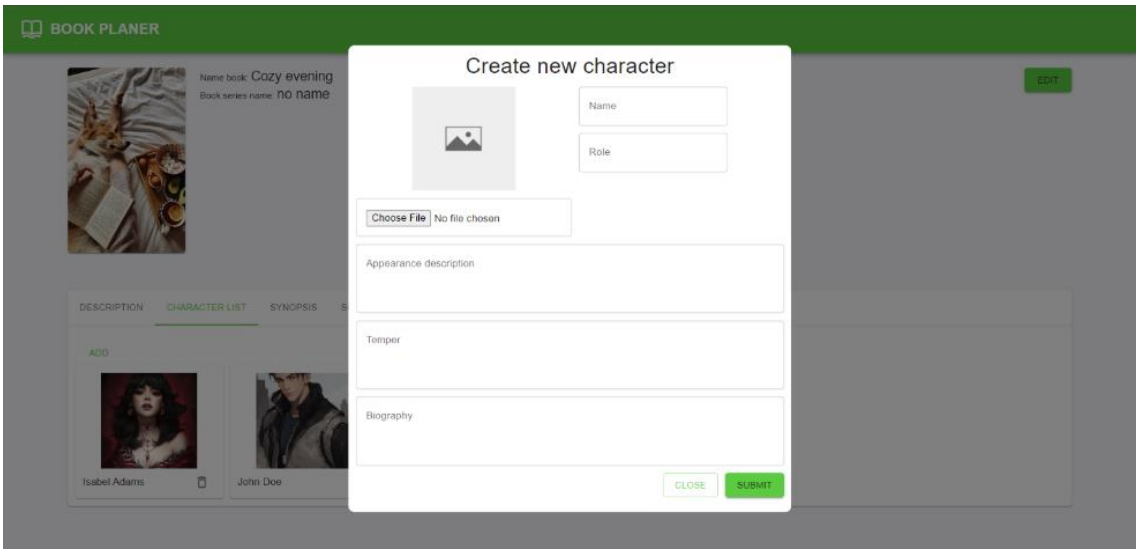


Рис. 3.4 Форма створення персонажу у додатку

Для відображення створених персонажів було створено компонент BookCharter. Він має у собі метод handleChangeCharacter, що відкриває вікно для редагування персонажа. А також, є методи, що відповідають за додавання нового персонажу до списку – addCharacter, внесення змін до існуючого персонажу списку – changeCharacter, та handleDelete, що відповідає за видалення персонажа зі списку. Існуючі персонажі відображаються під кнопкою ADD на сітці у вигляді невеликих карток, де кожен персонаж має зображення, ім’я, а також іконку для видалення.

Відображення створених персонажів на сторінці показано на рисунку 3.5.

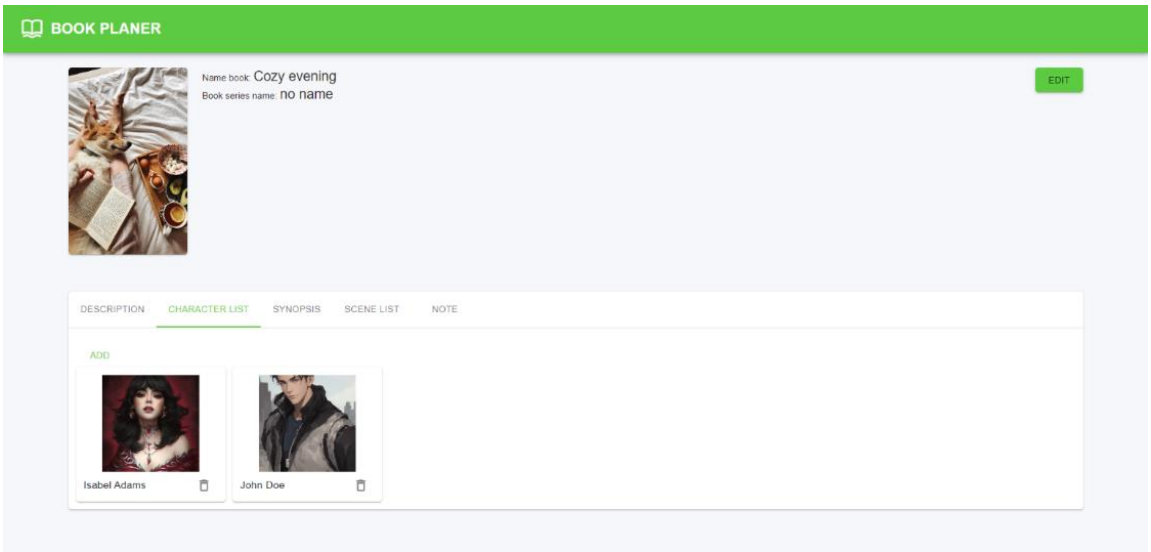


Рис. 3.5 Відображення персонажів у додатку

Для створення сцен реалізовано компонент BookSceneList. За створення сцени відповідає функція addScene, що додає нову сцену до списку сцен та закриває модальне вікно. Для редагування сцени створено метод handleChangeScene. Відповідно, для видалення було реалізовано метод handleDelete, що видаляє сцену зі списку за індексом. Створення відбувається шляхом заповнення форми, що має поля Name для надання назви сцени та полем Description, що буде зберігати опис сцени. На рисунку 3.6 показано реалізацію форми для створення сцени.

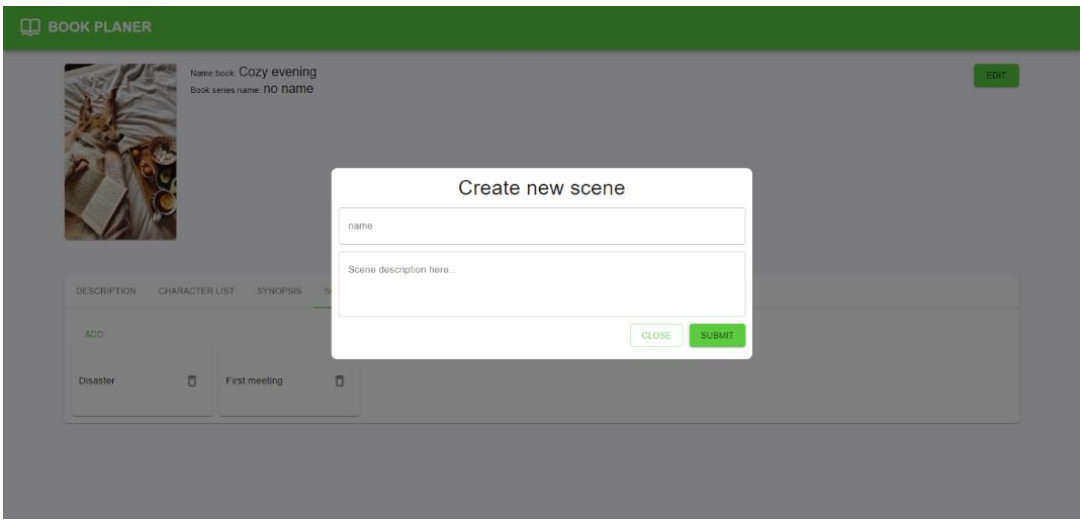


Рис.3.6 Форма для створення сцени у додатку

За відображення модального вікна відповідає компонент BookSceneModal. Для керування формою використовується useForm з бібліотеки react-hook-form. Для визначення чи редагується об’єкт є змінна isEdit. В якості обробки подій виступає метод onSubmit, який при значенні true для змінної isEdit викликає функцію onChangeScene, при значенні false – addScene. Вікно редагування сцени представлене на рисунку 3.7.

Останній компонент, що реалізується це BookNotes. Даний компонент відповідає за управління нотатками та їх відображенням на сторінці. Метод onSubmit відповідає за створення нового та редагування існуючого запису. handleChange відповідає за зміну стану компонента BookNotes, коли користувач

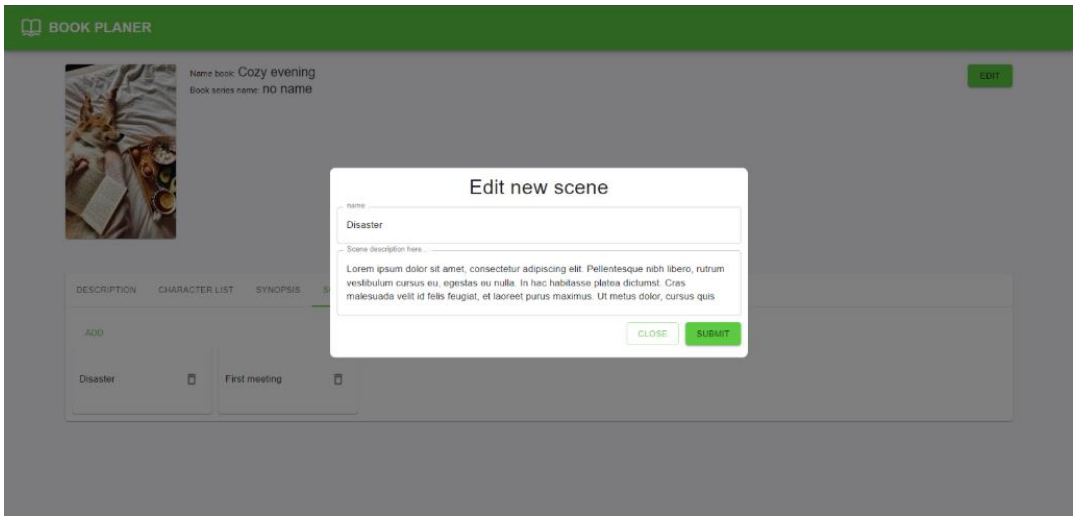


Рис.3.7 Форма редагування сцени

обирає редагування нотатки. Користувач для створення нотатки вводить текст у текстове поле та натискає кнопку Save. Після збереження, нотатка відображається під полем для введення, що відновилось до початкового стану, що представлене на рисунку 3.8.

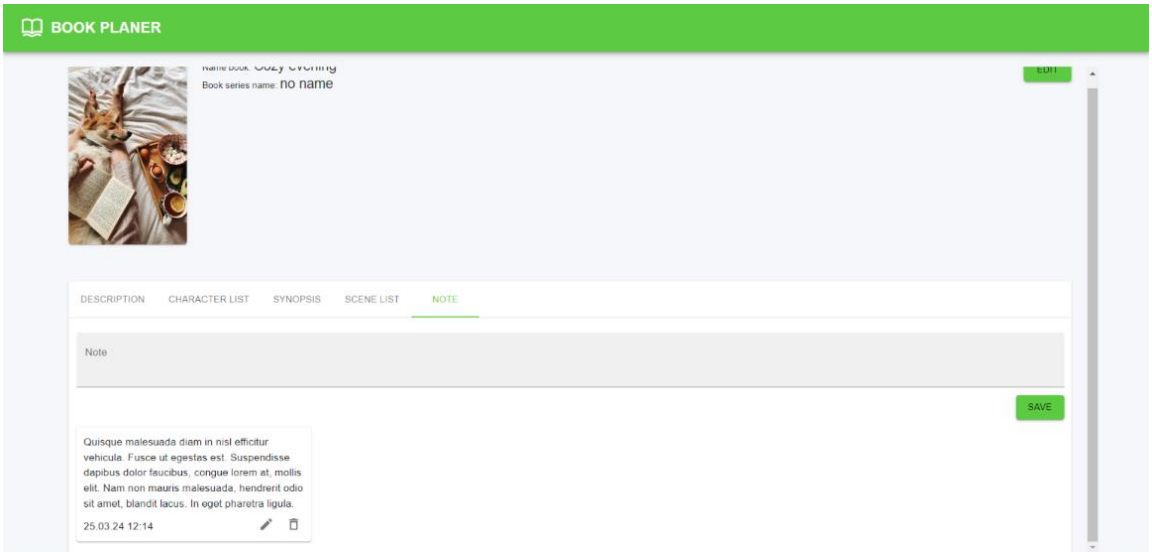


Рис.3.8 Відображення нотатків у додатку

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Актуальність тестування

Перед запуском web-застосунку та представлення його для користувача необхідно бути впевненим у якості та коректності роботи we-bзастосунку. Для визначення відповідності вимогам та правильного виконання ним функцій проводять тестування додатку.

За визначенням силлабусу ISTQB, тестування – це процес оцінки якості програмного забезпечення, що є частиною життєвого циклу розробки. Основними задачами тестування є пошук дефектів, переконання в якості продукту, запобігання виникнення помилок, оцінка робочих продуктів, перевірка на дотримання вимог, зменшення ризиків, перевірка на відповідність юридичним, договірним та нормативним вимогам та стандартам. [10]

Етап тестування починається ще під час визначення вимог. Тестування вимог дозволяє перевірити вимоги на:

- повноту опису функцій, які необхідно реалізувати;
- опис дійсно необхідного функціоналу;
- відсутність протиріч;
- однозначність.

Прототипування також є етапом тестування, оскільки створюючи модель майбутнього додатку можна перевірити вимоги на зручність використання інтерфейсу користувача.

Також, важливим є застосування різних видів тестування. Модульне тестування перевіряє роботу окремих компонентів на правильність роботи. Для перевірки взаємодії різних компонентів між собою застосовують інтеграційне тестування. Воно дозволяє виявити можливі проблеми та дефекти, що можуть вплинути на

неправильну роботу та взаємодію компонентів. Після інтеграційного тестування та перед приймальним виконується системне тестування. Даний етап перевіряє функції додатку як єдиного блоку в умовах наближених до реального використання. Під час системного тестування використовуються техніки тестування граничних значень, тестування з випадковими даними, вгадування помилок і головним є виконання регресійного тестування для перевірки, що внесені зміни не порушили роботу вже існуючих частин програми. Останнім етапом у життєвому циклі розробки є приймальне тестування. Основною метою даного етапу є перевірка всієї системи на відповідність вимогам та специфікаціям. Приймальне тестування може проводитись розробниками, тестувальниками компанії, або групою незалежних користувачів. Під час даного етапу виконують тестування функціональності, продуктивності, безпеки тощо.

Тестування web-застосунків перевіряє його на функціональність, безпеку, ефективність використання, адаптивність інтерфейсу під різні пристрої та інших аспектів для роботи через браузер в онлайн-середовищі.

Тестування функціональності web-застосунку виконує перевірку виконання функцій, що відповідають вимогам та очікуваним результатам. Перевіряється робота як окремих компонентів, так і всієї програми в цілому.

Тестування безпеки допомагає виявити уразливості системи на витік або втрату даних, системні збої. Перевірка аутентифікації та захист даних користувача.

Перевірка ефективності web-застосунку вимірює швидкість завантаження сторінок, час відповіді сервера для визначення продуктивності та відображення сторінок. Це допоможе забезпечити швидке та ефективне використання додатку для усіх користувачів.

Отже, тестування проводиться на кожному етапі розробки продукту для забезпечення якості роботи, надійності, безпеки. Також, це дозволяє виявити проблеми до запуску додатку та усунути їх без впливу на репутацію та фінансових збитків. Коректно проведений етап тестування дозволяє покращити

користувацький досвід та забезпечити конфіденційність даних. Тестування є критично важливим у розробці програмного забезпечення для якості користування та задоволення потреб юзерів.

4.2 Обґрунтування вибору підходів та технік тестування

Для тестування web-застосунку було застосовано декілька основних підходів та технік тестування. Для фази активного тестування вирішено використовувати мануальне тестування для більш якісної симуляції тестування та перевірки роботи додатку зі сторони користувача.

Позитивне та негативне тестування є одним з підходів, що перевіряє форми на правильність введених даних. Позитивна перевірка є важливою у етапі тестування, оскільки вона перевіряє правильність роботи системи при введенні коректних даних або при взаємодії користувача з програмою. Основною метою позитивного тестування є підтвердження, що програмне забезпечення функціонує правильно в умовах нормального використання. Під час перевірки вводяться правильні очікувані дані у форму для створення книги, персонажа та сцени, заповнюються усі поля. Також, це дозволяє перевірити правильність переходу між сторінками та категоріями меню. Позитивне тестування допомагає виявити помилки та проблеми, що можуть виникнути при нормальному використанні додатку. Позитивна перевірка є обов’язковою при тестуванні застосунків.

Негативною перевіркою тестується сценарій введення невірних даних у форми та перевірки неправильної поведінки системи. Основною метою негативного тестування є перевірка програмного забезпечення на реагування системи на введення неправильних даних у форми або відповідь системи на неочікувані дії зі сторони користувача. Під час цього етапу тестувальники вводять некоректні дані у поля форм або залишають порожніми поля, що мають бути обов’язково заповненими. Перевіряється реагування системи на негативні сценарії,

що не буде збережена форма персонажа без введення імені, форма сцени без назви, форма створення книги без введених назви книги та серії. За принципом про недосяжність вичерпного тестування, розуміємо, що повне тестування з усіма комбінаціями неможливе. Натомість, слід використовувати аналіз ризиків, різні методи тестування та пріоритизації перевірок функціональностей.[11]

Проактивний підхід тестування починається на ранніх стадіях розробки. Основною ідеєю є запобігання виникнення помилок ще на етапі проектування та визначення вимог. Тестувальники активно співпрацюють з розробниками ще на початкових етапах, аналізують вимоги на суперечливості, неводповідності, двозначності, створюють тест-кейси та тест-сценарії на основі базису. Аналіз архітектури продукту виявляє недоліки та проблеми в якості додатку. Проактивний підхід дає змогу зменшити ризики виникнення недоліків у продукті, що дозволить зменшити час на розробку та тестування.

Реактивний підхід – це стратегія, що передбачає знаходження та виправлення багів у додатку на пізніх етапах розробки або вже після випуску продукту. Даний підхід починається після завершення розробки та помилки знаходяться після введення додатку у користування. Помилки виправляють внесенням змін у код програми та знову перевіряють продукт на те, що баг був виправлений. Для отримання повідомлень про помилки, реактивний підхід спирається на отримання фідбеку від користувачів, юзер сторіз та реагують на них випуском патчів та оновлень. Реактивне тестування є менш ефективним ніж проактивне, оскільки виправляє проблеми вже після їх виникнення, а не до. Однак, даний підхід є важливим етапом для забезпечення користувачам максимально позитивний досвід користування.

Для проведення тестування вирішено використовувати техніки чорного ящика – це тестування програмного забезпечення при якому тестувальники не мають доступу до бази даних, не знають внутрішньої структури та як працює програма чи система та фокусуються на перевірці функціональності програми.[12]

Ще однією технікою, що використовуватиметься є техніка тестування засноване на чек-листах. Створюються та реалізуються тести, що покривають тестові умови, які прописані у чек-листі. Завдяки чек-листам систематизуються тестові сценарії та вимоги, що необхідно перевірити. Дана техніка дозволить провести тестування послідовно для кожного пункту чек-листа, що збільшить продуктивність та ефективність перевірки. Також чек-листи охоплюють широкий спектр аспектів програми такі як функціональність, безпека, ергономічність.

4.3 Результати тестування

Етап тестування почався з перевірки вимог. Для цього перевірено, що у вимогах повністю та чітко описана кожна функція та як вона працює, до чого відноситься, в якому вигляді має бути реалізована. Кожна функція зазначена у вимогах є необхідною та можливою для реалізації. У кожній формі є обов'язкові поля для заповнення, щоб дані були збережені. При описі роботи системи відсутні протиріччя, кожна категорія має лише одне призначення та не суперечить іншим вимогам.

Для перевірки роботи програми створено тест кейси на основі чек-листа, в якому прописані усі необхідні перевірки для тестування. При створенні умов для перевірок використані позитивні та негативні тести для перевірки усіх можливих проблем з покроковим виконання, очікуваним та фактичним результатами. Тест кейси з перевірками представлені у таблиці 4.1.

Таблиця 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
1	Відкрити web-застосунок вперше	1. Перейти за посиланням на сторінку додатку.	Відкривається вікно браузера. З'являється повідомлення про необхідність реєстрації.	Відкривається вікно браузера. З'являється повідомлення про необхідність реєстрації.
2	Відкрити web-застосунок як авторизований користувач	1. Перейти за посиланням на сторінку додатку.	Відкривається вікно додатку, відображається головна сторінка.	Відкривається вікно додатку, відображається головна сторінка.
3	Створити книгу	1. Відкрити застосунок. 2. Натиснути на кнопку створення книги на головній сторінці. 3. Ввести назву книги у поле «Book name». 4. Ввести назву серії книги у поле «Book series name». 5. Завантажити зображення. 6. Натиснути на кнопку «SUBMIT».	Книга створена та відображається в якості карточки на головній сторінці.	Книга створена та відображається в якості карточки на головній сторінці.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
4	Додати опис до книги	1. Відкрити web-застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. У категорії Description ввести опис книги у текстове поле. 4. Натиснути кнопку «SAVE» для збереження.	Введений опис збережений та відображається у текстовому полі.	Введений опис збережений та відображається у текстовому полі.
5	Створити персонажа книги, заповнивши усі поля.	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти у категорію Character list. 4. Натиснути на кнопку ADD. 5. Заповнити поле Name для збереження імені персонажа.	Персонаж успішно створений, відображається в якості картки в категорії Character list.	Персонаж успішно створений, відображається в якості картки в категорії Character list.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		6. Заповнити поле Role для вказання ролі. 7. Заповнити поле Appearance description. 8. Заповнити поле Temper. 9. Заповнити поле Biography. 10. Завантажити зображення. 11. Натиснути кнопку «SUBMIT» для збереження.		
6	Створити персонажа книги без імені.	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти у категорію Character list. 4. Натиснути на кнопку ADD.	Персонаж не створений.	Персонаж не створений.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		5. Заповнити поле Role для вказання ролі. 6. Заповнити поле Appearance description. 7. Заповнити поле Temper. 8. Заповнити поле Biography. 9. Завантажити картинку. 10. Натиснути кнопку «SUBMIT» для збереження.		
7	Редагувати існуючого персонажа	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти у категорію Character list. 4. Натиснути на картку персонажа.	Дані персонажа успішно змінені.	Дані персонажа успішно змінені.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		5. Змінити необхідні поля у формі редагування персонажа. 6. Натиснути на кнопку «SUBMIT» для збереження.		
8	Видалити існуючого персонажа	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти у категорію Character list. 4. Натиснути на іконку смітника для видалення персонажа.	Персонаж успішно видалений.	Персонаж успішно видалений.
9	Додати синопсис книги	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти до категорії Synopsis.	Синопсис успішно збережений та відображається у текстовому полі.	Синопсис успішно збережений та відображається у текстовому полі.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		4. Ввести інформацію у текстове поле. 5. Натиснути кнопку «SAVE» для збереження.		
10	Створити нову сцену.	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти до категорії Scene list. 4. Натиснути на кнопку «ADD» 5. У формі створення заповнити поле Name для назви сцени. 6. Ввести опис у поле Scene description here… 7. Натиснути кнопку «Submit» для збереження.	Сцена успішно створена та відображається у вигляді картки у категорії Scene list.	Сцена успішно створена та відображається у вигляді картки у категорії Scene list.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		її картку на головній сторінці. 3. Перейти до категорії Scene list. 4. Натиснути на іконку смітника для видалення.		
12	Редагувати створену сцену	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти до категорії Scene list. 4. Натиснути на картку сцени. 5. Внести необхідні зміни у формі. 6. Натиснути кнопку «SUBMIT» для збереження.	Сцена успішно відредагована.	Сцена успішно відредагована.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
13	Додати нову нотатку.	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти до категорії Note. 4. Ввести тіло нотатки у текстове поле. 5. Натиснути на кнопку «SAVE» для збереження.	Нотатка успішно створена та відображається у вигляді картки під текстовим полем.	Нотатка успішно створена та відображається у вигляді картки під текстовим полем.
14	Редагувати існуючу нотатку	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Перейти до категорії Note. 4. Натиснути на іконку ручку для редагування нотатки.	Нотатка успішно відредагована.	Нотатка успішно відредагована.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
		5. Редагувати тіло нотатки у текстовому полі. 6. Натиснути кнопку «SAVE» для збереження.		
15	Відредагувати існуючу книгу.	1. Відкрити застосунок. 2. Відкрити сторінку книги, натиснувши на її картку на головній сторінці. 3. Натиснути на кнопку «EDIT». 4. Внести зміни у формі редагування книги. 5. Натиснути кнопку «SUBMIT».	Дані про книгу успішно змінені.	Дані про книгу успішно змінені.
16	Видалити існуючу книгу.	1. Відкрити застосунок. 2. Натиснути на іконку смітника для видалення.	Книга успішно видалена.	Книга успішно видалена.

Продовження таблиці 4.1

ID	Опис	Кроки	Очікуваний результат	Фактичний результат
17	Створити книгу без назви	1. Відкрити застосунок. 2. Натиснути на кнопку створення книги. 3. Ввести назву серії книги. 4. Натиснути кнопку «SUBMIT»	Книга не створюється.	Книга не створюється.

Відповідно до результатів проведеного тестування, всі функції перевірені та працюють правильно. Інтерфейс користувача виконано відповідно прототипу та вимог. Під час тестування не виявлено серйозних помилок. Web-застосунок готовий до випуску та представлення кінцевим користувачам.

ВИСНОК

Під час виконання дипломної роботи було розроблено web-застосунку для планування книг «Book planer». У процесі виконання проведений аналіз актуальності книги в українському суспільстві, що спирався на результати дослідження Info Sapience.

Проаналізовані аналоги для планування книг: Stack Edit, Simplenote, Google Docs. Визначено переваги та недоліки кожного програмного рішення. Складено нефункціональні та функціональні вимоги до web-застосунку.

Обрано інструменти для реалізації web-застосунку, що написаний мовою програмування JavaScript з використанням бібліотеки React, а також мовою розмітки HTML та мовою стилів CSS. Для створення прототипу використовувався онлайн-сервіс Figma. Написання коду відбувалось у середі розробки WebStorm.

Під час моделювання web-застосунку, створено діаграму використання, для відображення взаємодії користувача з додатком та реакції системи на дії користувача. Окрім цього, спроектовано діаграму послідовності використання програми для відображення роботи системи.

Реалізовано web-застосунок відповідно поставленим функціональним та нефункціональним вимогам, що спрощує процес планування книги. Реалізований інтерфейс підтримується різними пристроями, такими як: смартфони, планшети, ПК. Web-застосунок відкривається через більшість популярних браузерів, що підтримують JavaScript.

Проведено тестування функціональності та відповідності вимогам. Усі виявлені дефекти були виправлені. Інтерфейс відображає усі елементи коректно на будь-якому пристрої та відповідає прототипу. Функції правильно реагують на дії користувача та виконують свою задачу. Модульне тестування підтвердило, що усі компоненти програми належним чином взаємодіють між собою.

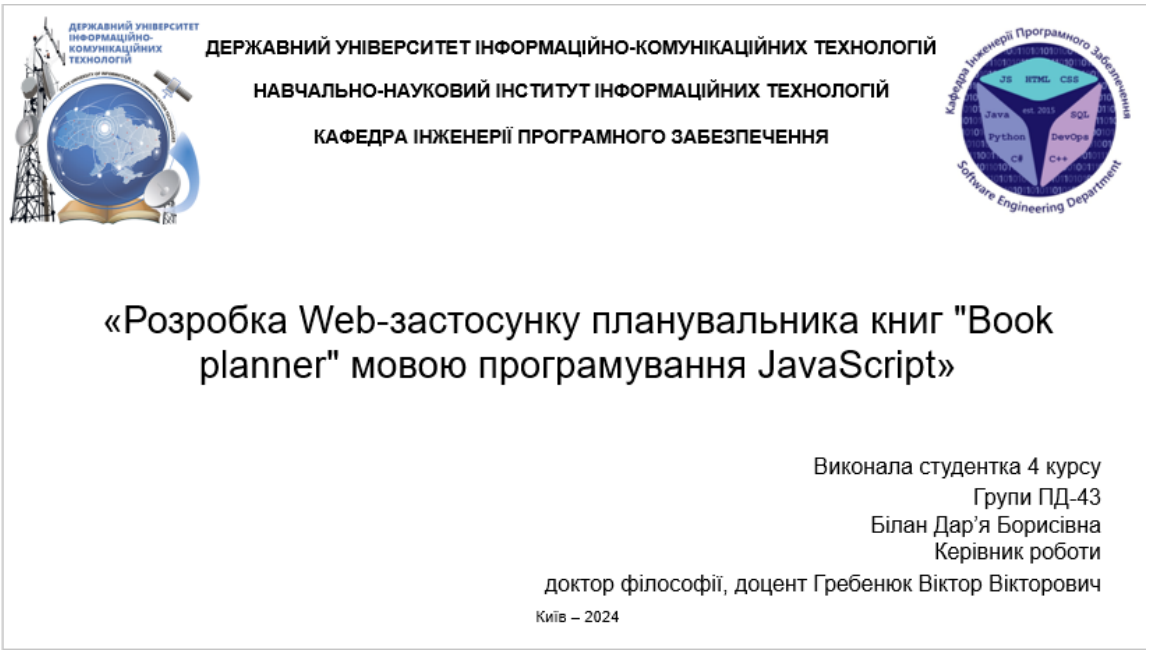
Диплом пройшов апробацію на наступних конференціях:

1. Білан Д. Б. Існуючі рішення для планування книг/ Негоденко О.В., Білан Д.Б.// Сучасний стан та перспективи розвитку IoT: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м. Київ – К.: Подано до друку
2. Білан Д. Б. Існуючі рішення для планування книг/ Негоденко О.В., Білан Д.Б.// Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ – К.: ДУІКТ, 2024, с. 103-105.

ПЕРЕЛІК ПОСИЛАНЬ

1. Волосевич І., Шуренкова А. Звіт за результатами всеукраїнського соціологічного дослідження «Читання в контексті медіаспоживання та життєконструювання». Київ, 2020. 181 с. URL: https://www.sapiens.com.ua/publications/socpol-research/148/Звіт_читання.pdf.
2. Офіційний сайт Ukrainian Reading and Publishing Data 2018. Ukrainian Reading and Publishing Data 2018. URL: https://data.chytomo.com/chytannya-v-ukrayini/
3. Robbins, Jennifer Niederst. Learning web design: A beginner's guide to HTML, CSS, JavaScript, and Web Graphics. " O'Reilly Media, Inc.", 2018.
4. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 4th ed. No Starch Press, 2024.
5. Felke-Morris T. Web Development and Design Foundations with HTML5. Pearson Education, Limited, 2020.
6. Pilgrim, Mark. HTML5: up and running: dive into the future of web development. " O'Reilly Media, Inc.", 2010.
7. Henick, Ben. HTML & CSS: The Good Parts: Better Ways to Build Websites That Work. " O'Reilly Media, Inc.", 2010.
8. Bertoli, Michele. React Design Patterns and Best Practices. Packt Publishing Ltd, 2017.
9. React Documentation. URL: https://legacy.reactjs.org/docs/getting-started.html.
10. ISTQB Glossary, V4.3. URL: https://glossary.istqb.org/en_US/search?term=&exact_matches_first=true.
11. Brief review on different manual software testing approaches & procedure. URL: https://pnrjournal.com/index.php/home/article/view/6707/8700.
12. UML Modeling and Black Box Testing Methods in the School Payment Information System. URL: https://iocscience.org/ejournal/index.php/mantik/article/view/969/671.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу планування книги за допомогою Web-застосунку мовою JavaScript.

Об'єкт дослідження: процес планування книги.

Предмет дослідження: Web-застосунок для планування книг.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз предметної галузі з планування книг.
2. Провести аналіз існуючих рішень для планування книг.
3. Визначення необхідних функціональних та нефункціональних вимог до Web-застосунку планування книг.
4. Провести аналіз інструментів для реалізації Web-застосунку для планування книг.
5. Розробити прототип Web-застосунку для планування книг.
6. Розробка Web-застосунку для планування романів «Book planner».
7. Тестування Web-застосунку.

3

АНАЛІЗ АНАЛОГІВ

Назви аналогів / Особливості	Stack Edit	Simplenote	Google Docs	Book Planer
Збереження інформації	кеш браузера та можуть бути втрачені	хмарне середовище Automattic	Google drive (15ГБ)	хмарне середовище Firebase Firestore (до 1 ГБ безкоштовної версії)
Структуризація інформації відповідно до певної книги	Налаштовується власноруч	Налаштовуються власноруч через додавання тегів	-	У додатку існує готова структура для планування книги
Наявність готових шаблонів для форми створення книги	-	-	-	+
Наявність готових шаблонів для форми створення персонажу	-	-	- (можливо використання готових шаблонів для резюме)	+
Наявність готових шаблонів для форми створення сцени	-	-	-	+
Доступ через браузери	Chrome, Firefox, Opera, Safari	-	Chrome, Firefox, Opera, Safari	Chrome, Firefox, Opera, Safari
Прив'язка зображень до книги та персонажа	-	-	+ (окрім svg, gif, tiff)	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Авторизація користувача за допомогою Google аккаунту.
2. Створення, редагування, видалення книги.
3. Створення, редагування, видалення персонажів сцен.
4. Створення, редагування, видалення сцен.
5. Створення, редагування, видалення нотатки.
6. Створення, редагування Description.
7. Створення, редагування Synopsis.

Нефункціональні вимоги

1. Мінімалістичний дизайн інтерфейсу з використанням стандартних іконок для взаємодії, для забезпечення використання застосунку користувачем, без необхідності навчання.
2. Кросплатформеність.
3. Кросбраузерність.
4. Середня швидкість обробки запитів у web-застосунку 20 ms.
5. Розробити наступні категорії до кожної книги: Description – опис книги, Characters list – список персонажів книги, Synopsis – короткий опис сюжету книги, Scene list – список сцен, Note – список нотатків.
6. Реалізувати створення та редагування книги, персонажа та сцени через форми, що мають структуру шаблону для створення відповідно кожного з об'єктів.

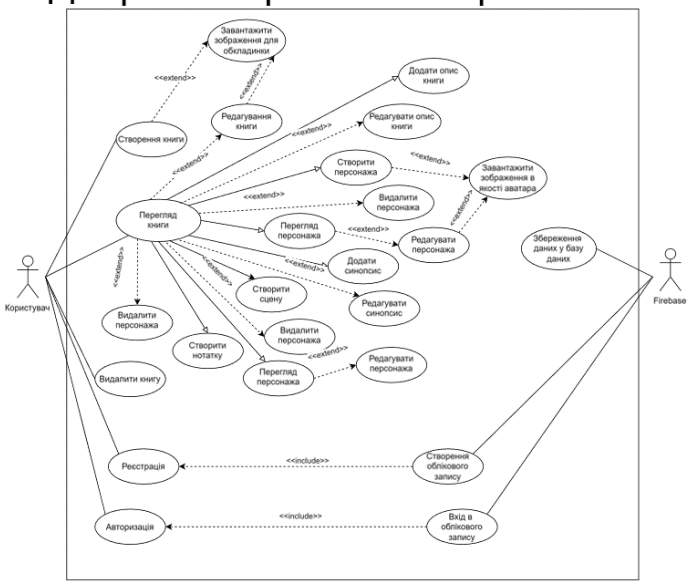
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Firebase

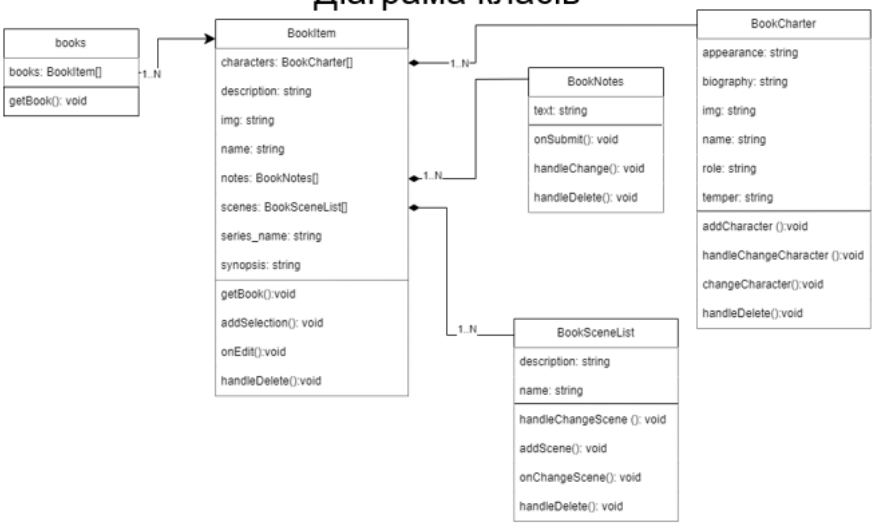
6

Діаграма варіантів використання



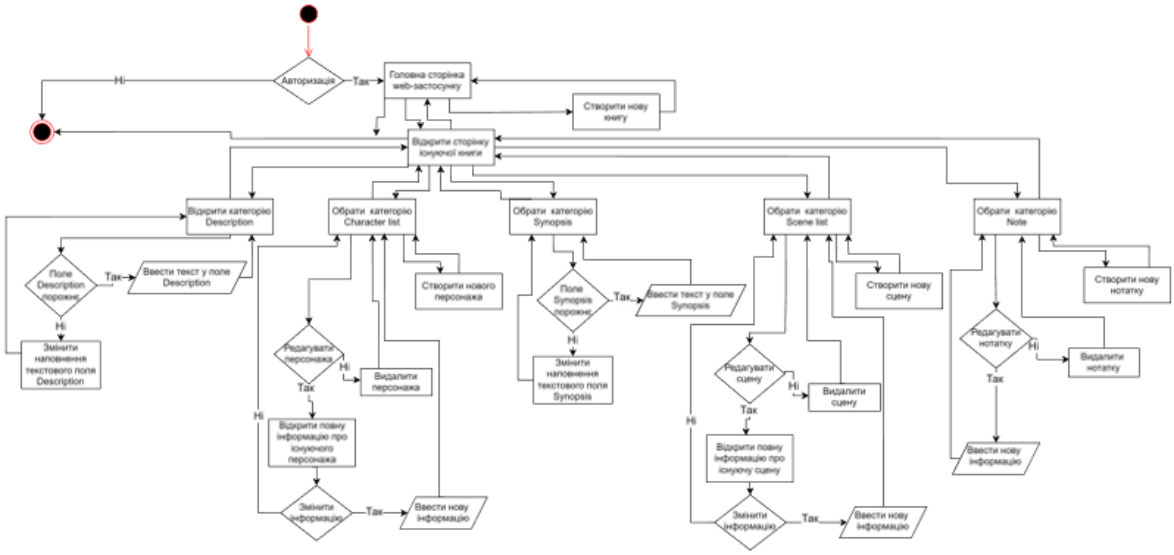
7

Діаграма класів



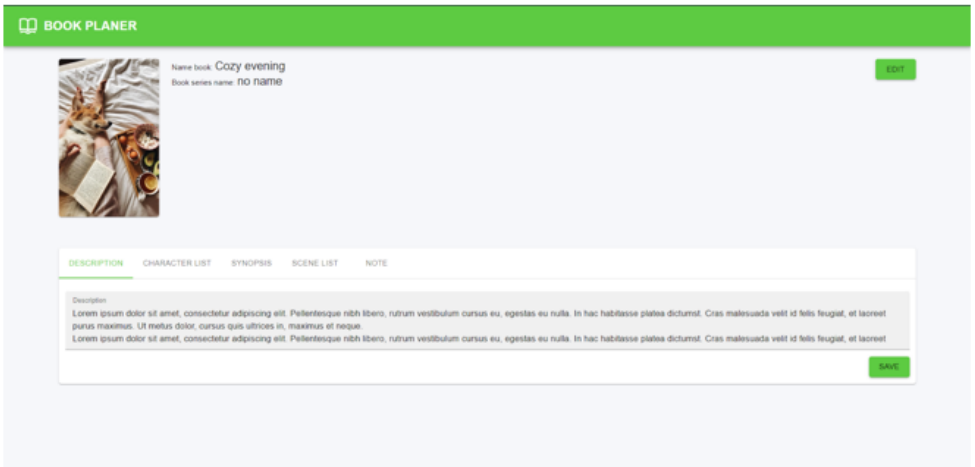
8

Схема роботи Web-застосунку



9

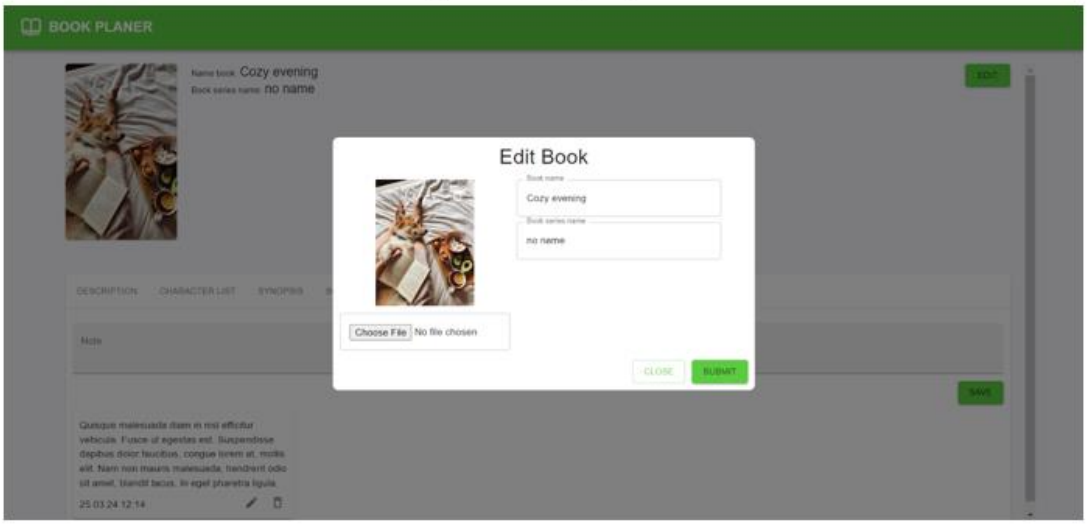
Екранна форма



Сторінка книги, категорія Description

10

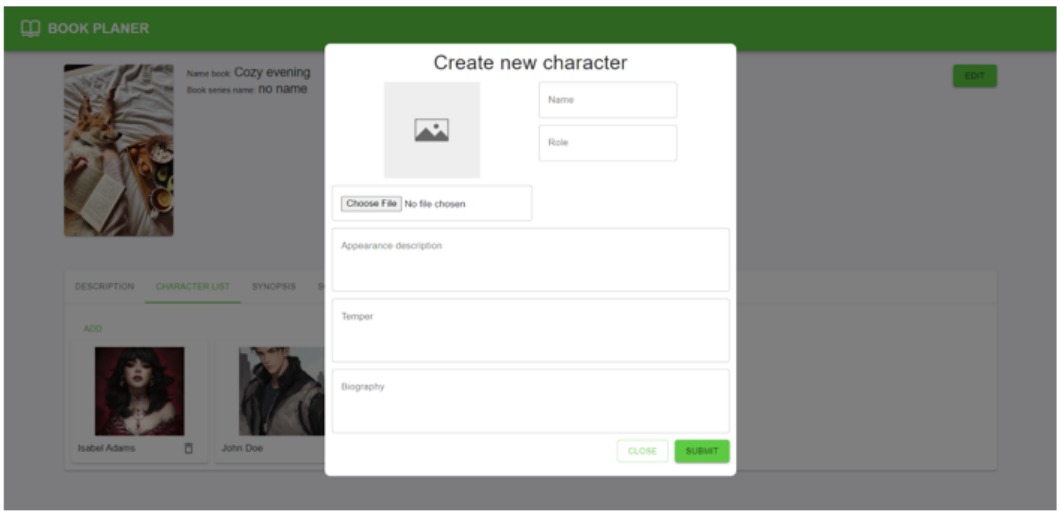
Екранна форма



Форма редагування книги

11

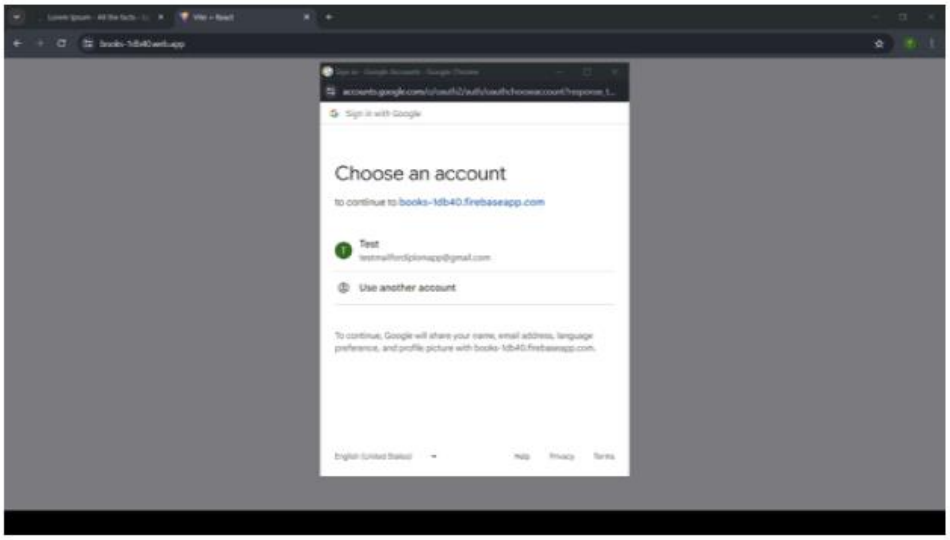
Екранна форма



Форма створення персонажа

12

Відео демонстрація роботи web-застосунку



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Білан Д. Б. Існуючі рішення для планування книг/ Негоденко О.В., Білан Д.Б.// Сучасний стан та перспективи розвитку IoT: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м. Київ. К.:
2. Білан Д. Б. Існуючі рішення для планування книг/ Негоденко О.В., Білан Д.Б.// Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024, с. 103-105

14

ВИСНОВКИ

1. Проаналізовано предметну галузь планування книг.
2. Проведено аналіз існуючих програмних рішень для планування книг.
3. Визначено основні функції, що мають бути реалізовані у Web-застосунку.
4. Обрано інструменти розробки для реалізації Web-застосунку. Застосунок реалізовувався у середі розробки Webstorm. Код програми писався на мові JavaScript з використанням бібліотеки React.js.
5. Створено прототип Web-застосунку за допомогою інструменту для прототипування Figma.
6. Розроблено Web-застосунок для планування книг «Book planner». Розроблено можливість створення об’єкту книги. До кожної книги створено категорії Description для опису, Character list для створення персонажів книги, Synopsis для короткого опису сюжету книг, Scene list для списку сцен у книзі, Note для нотатків.
7. Протестовано Web-застосунок на відповідність вимогам та правильність роботи програми. За результатами дослідження, Web-застосунок працює відповідно вимогам, без помилок.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

1. AuthProvider

```
import React, { useEffect, useState } from 'react';
// eslint-disable-next-line import/no-extraneous-dependencies
import { getAuth, signInWithPopup } from 'firebase/auth';

import { firebaseApp, googleAuthProvider } from '../../firebase';
import Loader from '../components/Loader/Loade';

const AuthProvider = ({ children }) => {
  const auth = getAuth(firebaseApp);
  const [user, setUser] = useState(auth.currentUser);

  useEffect(() => {
    auth.onAuthStateChanged((maybeUser) => {
      if (maybeUser !== null) {
        setUser(maybeUser);
      } else {
        signInWithPopup(auth, googleAuthProvider)
          .then((credentials) => setUser(credentials))
          .catch((e) => console.log(e));
      }
    });
```

```
  }, [auth]);

  return user === null ? <Loader /> : children;

};

export default AuthProvider;
```

2. AddBookModal

```
import React, { useEffect } from 'react';

import { Box, Button, Card, Grid, IconButton, TextField, Typography } from '@mui/material';

import { Controller, useForm, useWatch } from 'react-hook-form';

import { collection, addDoc } from 'firebase/firestore';

import { getAuth } from 'firebase/auth';

import moment from 'moment';

import DeleteOutlineIcon from '@mui/icons-material/DeleteOutline';

import MuiModal from '../../Modal';

import { firebaseApp, firebaseFirestore } from '../../../../firebase';

import useFileUpload from '../../../hooks/fileUpload';

import { useCommonContext } from '../../../context/commonContex.jsx';


const AddBookModal = ({ open = true, onClose, callback, isEdit, editData, onEdit }) => {

  const {

    control,
```

```
  handleSubmit,
  formState: { errors },
  setValue,
  reset
} = useForm();
const auth = getAuth(firebaseApp);
const { changeLoading } = useCommonContext();

const img = useWatch({ control, name: 'img' });

const { uploadFile, deleteFile  } = useFileUpload();

const handleAddFile = (file) => {
  changeLoading(true);
  uploadFile(file)
    .then((res) => {
      setValue('img', res);
      changeLoading(false);
    })
    .finally(() => {
      changeLoading(false);
    });
```

```
};


const onSubmit = async (data) => {
  try {
    if (isEdit) {
      onEdit(data);
      return;
    }

    await addDoc(collection(firebaseFirestore, 'books'), {
      ...data,
      userId: auth?.currentUser?.uid,
      created: moment().format('DD/MM/YYYY hh:ss:mm')
    });
    callback?.();
    onClose();
    reset();
    // TODO: add toast
  } catch (e) {
    console.error('Error adding document: ', e);
  }
};
```

```
const handleDeleteImg = () => {
  deleteFile(img?.path).then(() => reset({ img: undefined }));
};

useEffect(() => {
  if (isEdit) {
    reset(editData);
  }
}, [open]);

return (
  <MuiModal open={open}>
    <Box
      height="340px"
      display="flex"
      flexDirection="column"
      gap="10px"
      component="form"
      onSubmit={handleSubmit(onSubmit)}>
      <Box display="flex" alingItems="center" justifyContent="center">
        <Typography variant="h4" component="span" align="center">
```

```
          {isEdit ? 'Edit' : 'Add'} Book
        </Typography>
      </Box>
      <Grid columnGap="10px" height="100%" alignContent="center" container>
        <Grid alignItems="center" justifyContent="center" item xs={5}>
          <Box display="flex" flexDirection="column" alignItems="center" gap="10px">
            <Box
              sx={{
                height: '180px',
                width: '140px',
                display: 'flex',
                alignItems: 'center',
                justifyContent: 'center',
                position: 'relative',
                img: {
                  objectFit: 'cover',
                  objectPosition: 'center',
                  maxHeight: '180px',
                  height: '100%',
                  maxWidth: '140px',
                  width: '100%'
```

```
        }
      }}>
      {img?.src && (
        <Box position="absolute" zIndex={100} top="5px" right="5px">
          <IconButton
            w="20px"
            h="20px"
            onClick={(e) => {
              e.preventDefault();
              e.stopPropagation();
              handleDeleteImg();
            }}>
            <DeleteOutlineIcon sx={{ color: 'white' }} />
          </IconButton>
        </Box>
      )}

      <img src={img?.src || '/noImg.jpg'} alt="book-img" />
    </Box>

    <TextField type="file" onChange={(e) =>
handleAddFile(e.target.files[0])} />
  </Box>
```

```
</Grid>
<Grid item xs={6}>
  <Grid container xs={12} rowGap="8px">
    <Grid item xs={12}>
      <Controller
        name="name"
        control={control}
        render={({ field, fieldState }) => (
          <TextField
            {...field}
            fullWidth
            label="Book name"
            isError={!!fieldState?.error?.message}
            errorMessage={fieldState?.error?.message}
          />
        )}
      />
    </Grid>
    <Grid item xs={12}>
      <Controller
        name="series_name"
        control={control}
```

```
            render={({ field, fieldState }) => (
              <TextField
                {...field}
                fullWidth
                label="Book series name"
                isError={!!fieldState?.error?.message}
                errorMessage={fieldState?.error?.message}
              />
            )}
          />
        </Grid>
      </Grid>
    </Grid>
  </Grid>
  <Box display="flex" flexDirection="row" gap="10px" justifyContent="flex-end">
    <Button variant="outlined" onClick={onClose}>
      Close
    </Button>
    <Button type="submit" variant="contained">
      Submit
    </Button>
  </Box>
```

```
      </Box>
    </MuiModal>
  );
};
export default AddBookModal;
```

3. BookCharterModal

```
import React, { useEffect, useMemo, useState } from 'react';
import { Box, Button, IconButton, TextField, Typography } from
'@mui/material';
import { Controller, useForm, useWatch } from 'react-hook-form';
import DeleteOutlineIcon from '@mui/icons-material/DeleteOutline.js';
import MuiModal from '../../Modal';
import useFileUpload from '../../../hooks/fileUpload.jsx';
import { useCommonContext } from '../../../context/commonContex.jsx';

const BookCharterModal = ({ open, onClose, addCharacter, editItemState,
changeCharacter }) => {
  const {
    control,
    handleSubmit,
    formState: { errors },
    reset,
```

```
  setValue
} = useForm();
const { uploadFile, deleteFile } = useFileUpload();
const img = useWatch({ control, name: 'img' });
const { changeLoading } = useCommonContext();

const initialState = {
  name: '',
  role: '',
  appearance: '',
  temper: '',
  biography: '',
  img: { src: '', path: '' }
};

const isEdit = useMemo(() => {
  return editItemState !== null;
}, [editItemState]);

const handleClose = () => {
  reset({ initialState });
  onClose();
```

```
};

const onSubmit = (data) => {
  if (isEdit) {
    changeCharacter(data);
    reset(initialState);
    return;
  }
  addCharacter(data);
  reset(initialState);
};

useEffect(() => {
  if (isEdit) {
    reset(editItemState);
    // console.log(editItemState);
  }

  if (open === false) {
    reset();
  }
}, [open]);
```

```
const addImage = (file) => {
  changeLoading(true);
  uploadFile(file)
    .then((res) => {
      changeLoading(false);
      setValue('img', res);
    })
    .finally(() => {
      changeLoading(false);
    });
};
const removeImage = () => {
  deleteFile(img?.path).then(() => setValue('img', { src: '', path: '' }));
};

return (
  <MuiModal open={open}>
    <Box
      height="610px"
      display="flex"
      flexDirection="column"
```

```
        gap="10px"
        component="form"
        onSubmit={handleSubmit(onSubmit)}>
        <Box display="flex" alingItems="center" justifyContent="center">
          <Typography variant="h4" component="span" align="center">
            {isEdit ? 'Edit' : 'Create new'} character
          </Typography>
        </Box>
        <Box display="flex" flexDirection="column" gap="10px">
          <Box display="flex" gap="10px">
            <Box display="flex" flexDirection="column" alignItems="center"
gap="10px">
              <Box
                sx={{
                  height: '140px',
                  width: '140px',
                  display: 'flex',
                  alignItems: 'center',
                  justifyContent: 'center',
                  position: 'relative',
                  img: {
                    objectFit: 'cover',
                    objectPosition: 'center',
```

```
    maxHeight: '140px',
    height: '100%',
    maxWidth: '140px',
    width: '100%'
  }
}}>
{img?.src && (
  <Box position="absolute" zIndex={100} top="5px" right="5px">
    <IconButton
      w="20px"
      h="20px"
      onClick={(e) => {
        e.preventDefault();
        e.stopPropagation();
        removeImage();
      }}>
      <DeleteOutlineIcon sx={{ color: 'white' }} />
    </IconButton>
  </Box>
)}

<img src={img?.src || '/noImg.jpg'} alt="book-img" />
```

```
          </Box>

          <TextField type="file" onChange={(e) => addImage(e.target.files[0])}
/>
        </Box>

        <Box display="flex" flexDirection="column" gap="10px">
          <Box>
            <Controller
              name="name"
              control={control}
              render={({ field, fieldState }) => (
                <TextField
                  {...field}
                  fullWidth
                  label="Name"
                  isError={!!fieldState?.error?.message}
                  errorMessage={fieldState?.error?.message}
                />
              )}
            />
          </Box>
          <Box>
```

```
        <Controller
          name="role"
          control={control}
          render={({ field, fieldState }) => (
            <TextField
              {...field}
              fullWidth
              label="Role"
              isError={!!fieldState?.error?.message}
              errorMessage={fieldState?.error?.message}
            />
          )}
        />
      </Box>
    </Box>
  </Box>
  <Box display="flex" flexDirection="column" gap="10px">
    <Box>
      <Controller
        name="appearance"
        control={control}
        render={({ field, fieldState }) => (
```

```
      <TextField
        {...field}
        multiline
        minRows={3}
        maxRows={3}
        fullWidth
        label="Appearance description"
        isError={!!fieldState?.error?.message}
        errorMessage={fieldState?.error?.message}
      />
    )}
  />
</Box>
<Box>
  <Controller
    name="temper"
    control={control}
    render={({ field, fieldState }) => (
      <TextField
        {...field}
        multiline
        minRows={3}
```

```
        maxRows={3}
        fullWidth
        label="Temper"
        isError={!!fieldState?.error?.message}
        errorMessage={fieldState?.error?.message}
      />
    )}
  />
</Box>
<Box>
  <Controller
    name="biography"
    control={control}
    render={({ field, fieldState }) => (
      <TextField
        {...field}
        multiline
        minRows={3}
        maxRows={3}
        fullWidth
        label="Biography"
        isError={!!fieldState?.error?.message}
```

```
                errorMessage={fieldState?.error?.message}
              />
            )}
          />
        </Box>
      </Box>
    </Box>
    <Box display="flex" flexDirection="row" gap="10px" justifyContent="flex-end">
      <Button variant="outlined" onClick={handleClose}>
        Close
      </Button>
      <Button type="submit" variant="contained">
        Submit
      </Button>
    </Box>
  </Box>
 </MuiModal>
 );
};
export default BookCharterModal;
```

4. BookSceneModal

```
import React, { useEffect, useMemo, useState } from 'react';
import { Box, Button, TextField, Typography } from '@mui/material';
import { Controller, useForm } from 'react-hook-form';
import MuiModal from '../../Modal';

const BookSceneModal = ({ open, onClose, addScene, editItemState,
onChangeScene }) => {
  const {
    control,
    handleSubmit,
    formState: { errors },
    reset
  } = useForm();

  const isEdit = useMemo(() => {
    return editItemState !== null;
  }, [editItemState]);

  const handleClose = () => {
    reset();
    onClose();
  };
```

```
const onSubmit = (data) => {
  if (isEdit) {
    onChangeScene(data);
    return;
  }
  addScene(data);
};

useEffect(() => {
  if (isEdit) {
    reset(editItemState);
  }

  if (open === false) {
    reset();
  }
}, [open]);

return (
  <MuiModal open={open}>
    <Box
```

```
height="252px"
display="flex"
flexDirection="column"
gap="10px"
component="form"
onSubmit={handleSubmit(onSubmit)}>
<Box display="flex" alingItems="center" justifyContent="center">
  <Typography variant="h4" component="span" align="center">
    {isEdit ? 'Edit' : 'Create'} new scene
  </Typography>
</Box>
<Box display="flex" flexDirection="column" gap="10px">
  <Box display="flex" flexDirection="column" gap="10px">
    <Box>
      <Controller
        name="name"
        control={control}
        render={({ field, fieldState }) => (
          <TextField
            {...field}
            fullWidth
            label="name"
```

```
        isError={!!fieldState?.error?.message}
        errorMessage={fieldState?.error?.message}
      />
    )}
  />
</Box>
<Box>
  <Controller
    name="description"
    control={control}
    render={({ field, fieldState }) => (
      <TextField
        {...field}
        multiline
        minRows={3}
        maxRows={3}
        fullWidth
        label="Scene description here..."
        isError={!!fieldState?.error?.message}
        errorMessage={fieldState?.error?.message}
      />
    )}
```

```
            />
          </Box>
        </Box>
      </Box>
      <Box display="flex" flexDirection="row" gap="10px" justifyContent="flex-
end">
        <Button variant="outlined" onClick={handleClose}>
          Close
        </Button>
        <Button type="submit" variant="contained">
          Submit
        </Button>
      </Box>
    </Box>
  </MuiModal>
 );
};
export default BookSceneModal;
```

5. Main page

```
import React, { useEffect, useState } from 'react';
import { Box, Fab, Grid } from '@mui/material';
import AddIcon from '@mui/icons-material/Add';
```

```
import { collection, getDocs, where, query, doc, deleteDoc } from
'firebase/firestore';
import { getAuth } from 'firebase/auth';
import { Link } from 'react-router-dom';
import BookItem from '../../components/BookItem';
import AddBookModal from '../../components/Modals/AddBookModal';
import { firebaseApp, firebaseFirestore } from '../../../firebase';
import { useCommonContext } from '../../context/commonContex';

const Main = () => {
  const [openModal, setOpenModal] = useState(false);
  const [books, setBooks] = useState([]);
  const auth = getAuth(firebaseApp);
  const { changeLoading } = useCommonContext();

  const getBooks = async () => {
    changeLoading(true);
    const querySnapshot = await getDocs(
      query(collection(firebaseFirestore, 'books'), where('userId', '==',
auth?.currentUser?.uid))
    );
    const data = [];
    querySnapshot.forEach((docEl) => {
```

```
    data.push({ ...docEl.data(), id: docEl.id });
  });
  setBooks(data);
  changeLoading(false);
};

const deleteBook = async (id) => {
  const docRef = doc(firebaseFirestore, 'books', id || '');
  await deleteDoc(docRef);
  getBooks();
};

useEffect(() => {
  getBooks();
}, []);

return (
  <Box mt="20px">
    <Grid container gap="20px">
      {books.map((el) => (
        <Grid item key={el.id}>
          <Box
```

```
        component={Link}
        sx={{
          color: 'inherit',
          textDecoration: 'inherit'
        }}
        to={`/book/${el.id}`}>
        <BookItem
          name={el.name}
          image={el.img?.src || ''}
          handleDelete={() => deleteBook(el.id)}
        />
      </Box>
    </Grid>
  ))}
</Grid>
<Fab
  onClick={() => setOpenModal((prev) => !prev)}
  sx={{ position: 'fixed', bottom: 20, right: 20 }}
  color="primary"
  aria-label="add">
  <AddIcon />
</Fab>
```

```
      <AddBookModal
        open={openModal}
        onClose={() => setOpenModal(false)}
        callback={() => getBooks()}
      />
    </Box>
  );
};
export default Main;
```