Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка веб додатку для букшерінгу

Текстова частина до курсової роботи за спеціальністю
«Прикладна математика» - 113

Керівник курсової роботи
Асистент
Калітовський Б.В
_____ (Підпис)
“___” _____ 2021 року
Виконав студент ПМ-3
Колінько П. В
“___” _____ 2021 року

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мульмедійних систем,

доц., к.ф.-м.н.

_____ Жежерун О. П.

(підпис)

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Колінько Павлу факультету інформатики 3 курсу

ТЕМА: Розробка веб додатку для букшерінгу

Вихідні дані:

- Веб-застосунок, розроблений з використанням React, Nest.js

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз існуючих рішень

2 Теоретичні відомості

3 Проектування та розробка додатку

Висновки

Дата видачі „___” _____ 2021 р. Керівник _____

Завдання отримав _____

Зразок календарного плану виконання курсової роботи

Тема: Розробка веб додатку для букшерінгу

Календарний план виконання роботи:

№	Назва етапу курсового проекту (роботи)	Термін виконання
1.	Отримання завдання на курсову роботу	листопад
2.	Аналіз рішень, огляд локументацій за темою курсової роботи	листопад-грудень
3.	Створення практичної частини роботи	січень-березень
4.	Написання текстової частини роботи	березень-квітень
6.	Надання роботи керівнику для перевірки	кінець квітня
8.	Коригування роботи за результатами перевірки	кінець квітня-початок травня
9	Подання роботи на кафедру для перевірки на плагіат	щонайменше за два тижні до захисту
10.	Захист курсової роботи	середина травня

Студент _____

Керівник _____

“_____” _____

ЗМІСТ

Анотація

У даній курсовій роботі розглянута проблема процесу обміну книгами між людьми, що зацікавленні у читанні паперової літератури. Було створено веб-застосунок для спрощення та централізації процесу обміну для більш комфортної роботи, та адміністрування даного процесу. Також, було створено алгоритм підбору рекомендації для кожного користувача.

ВСТУП

Література будь-якої тематики та жанрів є популярною завжди. Наразі, література в електронному форматі має такий самий обсяг, як і паперові книжки. Але навіть попри це, паперова література не втрачає своєї актуальності. Для однієї людини це питання зручності, для іншої – питання вподобань. Але така література коштує грошей і на відміну від електронної, не може бути отримана безкоштовно. Саме в таких випадках на допомогу приходить книгообмін, як можливість заощадити кошти та поділитись улюбленими творами.

У наш час існує чимало акції книгообміну, переважно у соціальних мережах, але були і ті, що відбувались просто неба, як декілька років тому, коли була започаткована акція книгообміну серед населення за допомогою «жовтих скриньок», що знаходились просто на вулиці. На жаль ця акція, на мою думку, не була грамотно реалізована.

Дуже часто, судячи з досліджень та опитувань, акція обміну книжками у соціальних мережах має певний перелік незручностей. Наприклад - децентралізованість, великий об’єм книг, за відправлення яких часто відповідають лише декілька людей, які організували обмін, фактична неможливість заборони участі тим, хто порушував правила. З цих та інших факторів нерідко випливають різні проблеми, наприклад – не отримання книжок або отримання не тією літератури, яку людина очікувала.

Мета моєї курсової роботи – створення веб-застосунку для спрощення та впорядкування процесу обміну книжками між зацікавленими людьми, та можливість адміністрування процесу.

Курсова робота складається з наступних розділів:

Мета розробку застосунку. У розділі проведено аналіз вже існуючих додатків або їх аналогів, акції книгообміну. З’ясовано основні проблеми, що виникають під час таких акцій та сформульовані умови для веб-застосунку.

Опис теоретичних відомостей. У розділі було описано кожну технологію, що було використано при розробці клієнту та серверу. Зазначено, чому були вибрані саме такі інструменти для розробки застосунку.

Опис процесу розробки додатку. Опис процесу розробки додатку включає у себе опис та обґрунтування прийнятих архітектурних рішень. Описаний процес розробки клієнту та серверу.

РОЗДІЛ 1. Аналіз існуючих рішень.

1.1 Мета

Обмін книжками – розповсюджене явище. Він допомагає економити гроші, дізнатися смаки в літературі інших людей та поділитися своїми.

Під час збору даних про акції книгообміну, частіше за все мені траплялися такі акції у соціальних мережах. Як показує практика, вони реалізовані за одним шаблоном. Створюється спеціальний аккаунт, наприклад в соціальній мережі Instagram, та починається масштабна піар-компанія. Якщо людна бажає прийняти участь у обміні, частіше за все, їй необхідно підписатись на аккаунт або залишити коментар під оголошенням. Коли спливає час акції, відбувається обмін контактами (хто і яку літературу отримує, які випадковим чином обирає адміністрація або люди, що реалізували акцію обміну).

Така схема має певний перелік труднощів. Я хотів би заначити наступні: обмеженість часу акції, складність для організаторів проведення акції, необхідність систематичного піару аккаунта та неможливість вибору цікавої саме для користувача літератури.

Обмеженість в часі. Частіше за все такі акції тривають декілька тижнів і не повторюються за певного інтервалу. Якщо людина не встигає прийняти участь, то невідомо, скільки їй необхідно буде чекати (або шукати нову) акцію.

Складність для організаторів. Насправді, достатньо складно реалізувати таку акцію для великої кількості користувачів. Необхідність піару та заохочення нових користувачів – лише дві поширені проблеми з ряду інших.

Неможливість персоналізації вибору. Як було зазначено, вибір «пар» для обміну книгами відбувається випадковим чином. Це добре з точки зору пізнання нових авторів та жанрів, але цілком можлива ситуація отримання тієї літератури, у якій людина не зацікавлена.

Також, я знаходив спільноту за схожою тематикою у інтернет порталі Pikabu. Але основна проблема, що цей сайт створений як блог під різні тематики, і за своїм призначенням не підходить для акції книгообміну.

1.2 Постановка задачі

Метою моєї курсової роботи є створення зручного та просто з точки зору користувача веб-застосунку з адаптивним дизайном.

Основні характеристики, які повинен включати у себе застосунок це:

- Адаптивний дизайн ;
- Інтуїтивно зрозумілий користувацький інтерфейс (або підказки за необхідністю) ;
- Можливість самостійного обміну між користувачами без участі адміністрації ;
- Можливість звернення до адміністрації для вирішення питань ;
- Алгоритм підбору рекомендації .

1.3 Висновки до розділу 1

У розділі «Аналіз предметної області» було досліджено схожі ресурси та акції онлайн обміну книжками. Були проведені опитування людей, що приймали участь у таких акціях, проаналізовані їх зауваження та побажання.

РОЗДІЛ 2. Теоретичні відомості

2.1 Рекомендаційні системи та алгоритми

Основна задача рекомендаційної системи або алгоритму – проінформувати користувача про товар або послугу, яка теоретично може йому сподобатись.

Велика кількість платформ, онлайн та мобільних застосунків використовують рекомендаційні алгоритми з метою зробити свій продукт більш привабливим. Spotify, ITunes, Netflix, YouTube – та багато інших ресурсів мають у своєму арсеналі різні алгоритми для підбору товарів або послуг. Частіше за все, алгоритми на великих ресурсах є дуже складними, використовують нейронні мережі задля досягнення мети – підбору рекомендацій, та не афішуються на публіку задля збереження конкурентоспроможності.

Але, існують загальні патерни для створення їх більш простого аналогу. Частіше за все, такий алгоритм буде мати наступні характеристики:

- Предмет рекомендації(товари або послуги, тощо) ;
- Мета рекомендації ;
- Джерело рекомендації(данні користувача, аудиторія сайту, тощо) ;
- Рівень персоналізації ;
- Прозорість ;
- Актуальність .

Особливу увагу треба приділити 3 останнім пунктам – рівень персоналізації, прозорість, актуальність [9].

Рівень персоналізації. Неперсоналізовані рекомендації – це рекомендації, які засновані на тезисі «Сподобалось всім, і вам сподобається».

Іншими словами, такі алгоритми пропонують всім людям одні й ті самі продукти, враховуючи вибір кожного. Персоналізовані рекомендації – це рекомендації, що на відміну відвід неперсоналізованих, засновані на вашому виборі. Але що робити, якщо користувач тільки но зареєструвався на сайті? Що йому пропонувати в такому випадку? Тут на допомогу приходить згладжене середнє (Damped mean) – це підхід коли використовують не лише дані конкретного користувача, а також задіяні середні показники всіх користувачів.

Прозорість. Прозорість – це яким саме чином були отримані дані для подальшого надання рекомендацій. Один з способів це умовне віконце, де буде розташоване невелике опитування, після проходження якого користувачем система отримує нову інформацію для надання рекомендацій. Інший спосіб – це неявний збір даних, своєрідне таємне спостереження за користувачем. Як приклад, відслідковування, що він додав в кошик або улюблені, тощо.

Актуальність. При наданні рекомендацій, актуальність - це дуже важливе питання. Зрозуміло, що те, що подобалось користувачу декілька місяців або тим більше років назад може не сподобатись йому зараз. Через це старі дані відсіюють за марністю. Частіше за все для цього використовують формули або алгоритми, які є індивідуальними під кожну конкретну тематику продукції.

2.2 Використані технології та інструменти

У розділі описані інструменти та технології з допомогою яких розроблявся веб-застосунок.

2.2.1 Клієнт

Клієнт – це частина веб-застосунку, що відображається у браузері. Наразі, клієнтська частина розробляться у декілька етапів, а саме – розробка дизайну та

макету, перенесення дизайн в компоненти, додання логіку та інтерактивності й інтеграція написаного API.

2.2.1.1 React

React – це один з трьох (Angular, View, React) найпопулярніших UI бібліотек у світі, створено компанією Facebook, як конкурент для Angular від Google. UI(User interface) бібліотека – це бібліотека спрямована на розробку користувацьких інтерфейсів. Іншими словами, розробка візуальної складової веб-застосунку. React у свої основі використовує компоненто-орієнтований підхід [1].

React найчастіше за все використовують для розробки додатків малого, середнього та більше середнього розмірів. Його переваги над іншими фреймворками:

- швидкість роботи самого React ;
- простота ;
- низький поріг входу .

Два головних його недоліки – це відсутність підтримки різних технології «з коробки», та відсутність чіткої структури. Перша проблема означає, що буде необхідно встановлювати велику кількість додаткових залежностей. Друга проблема сильніше за все відчувається при розробці додатків розміром більше за середній.

2.2.1.2 Antd-design

Хоча бібліотека React надає дуже зручні інструменти для розробки, вона містить далеко не все. Одна з проблем при створенні інтерфейсу, це необхідність його покрокового створення власними руками. Але є дуже зручні інструменти для розробки дизайну то невеликих шматків бізнес-логіки сайту.

Одне розповсюджених рішень – це UI бібліотеки. Bootstrap, material UI та інші допомагають сильно економити час при створенні дизайну веб-застосунку.

Дуже популярним та зручним представником є Antd-design. Це бібліотека, що містить готові компоненти для створення власного дизайн. Вона використовує підхід стилізованих компонентів(Styled components), що дуже спрощує роботу.

Маючи гарно написану документацію, простоту «кастомізації» та добре описані випадки коли необхідно використовувати той чи інший варіант робить бібліотеку Antd-design найкращим рішенням для створення дизайну сайту.

2.2.1.3 Typescript

Javascript – мультипарадигменна мова програмування. Вона підтримує об’єктно орієнтовані, імперативні та функціональні стилі програмування. Мова програмування javascript – універсальний та зручний інструмент для створення сайтів. На ній можливо з легкістю розробити клієнтську та серверну частини веб-додатку [4]. Основні характеристики які притаманні цій мові програмування:

- динамічна типізація ;
- слабка типізація ;
- автоматичний контроль над пам’яттю ;
- прототипне наслідування .

Динамічна типізація – це прийом у мовах програмування коли тип змінній надається не під час оголошення змінної, а під час присвоювання їй значення.

Наприклад Python та Javascript підтримують її, на відміну наприклад від мови програмування Java або C++.

Такий спосіб має свої переваги та недоліки. Одна з переваг - це простота використання. Немає необхідності чітко звіряти типи змінних при присвоюванні.

За необхідністю, можливо змінній, що мала чисельний тип даних присвоїти строковий тип даних, що дозволяє не створювати велику кількість змінних.

Але, вище зазначені переваги також можуть бути недоліками. Під час написання дуже великих проектів легко заплутатись у типах даних, що неодмінно призведе до проблем. У цьому випадку використовується Typescript.

Typescript – це мова програмування, що по суті розширює можливості Javascript. Вона була розроблена компанією Microsoft у 2012 році. Він є зворотньо сумісний з JS та компілюється в останній.

Основна відмінність полягає у можливості явної типізації даних та підтримка об’єктно-орієнтованого програмування як у класичних ООП мовах.

Всі зазначені вище відмінності дозволяють розробляти веб-застосунки та не заплутатись у типах. Також, застосунки створені за допомогою Typescript легше підтримувати іншим розробникам проекту.

2.2.1.4 Redux

Redux – це бібліотека, розробка якої ведеться у відкритому репозиторії Github. Використовується для розробки клієнтської частини з метою контролю «станами» веб-застосунку. Частіше за все, використовується разом з React, хоча по факту може бути використана не тільки з ним, а як і незалежна бібліотека. Для роботи з вищезазначеною бібліотекою використовується інструмент React-redux.

Яку саме проблему допомагає вирішити Redux? Всі хто розробляв додатки за допомогою React однозначно стикалися з проблемою обміну даними між компонентами. Як приклад, візьмемо типову ситуацію – авторизація користувача на сайті. Дані користувача надходять з сервера на одній сторінці, а передати їх необхідно в особистий кабінет на зовсім іншій сторінці. До створення бібліотеки

Redux використовувався підіхід передачі даних від батьківського компонента до дочірнього за допомогою так званих «пропсів», а передача вже оброблених даних від дочірнього до батьківського компонента здійснювалось шляхом передачі через callback-функцію та подальшого присвоювання даних у батьківському компоненті.

Такий підхід є не зручним та не гнучким у розробці, оскільки дуже легко заплутатись. Redux вирішує цю проблему, шляхом створення локального сховища, у якому, зберігаються всі необхідні дані, а за допомогою бібліотеки React-redux та двох хуків, що створені у ній, до цих самих даних можливо отримати доступ у будь якому компоненті.

2.2.1.5 Redux-thunk

На жаль, бібліотека Redux не спроможна вирішити всі проблеми з якими стикаються під час розробки застосувань.

Класична модель роботи з бібліотекою Redux . Необхідно зробити функцію, яку називають reducer. У неї передається початковий стан – це об’єкт, який характеризує тип подальших даних, від початку частіше за все є порожнім[2].

Функція reducer має єдину мету – модифікувати та записувати ті дані, що у неї приходять. Як приклад, збереження масиву, або зміна поля конкретного елементу у масиві шляхом ітерації, тощо. Також використовують так звані action-функції. Їх мета передати в reducer-функції дані з якими необхідно щось зробити та тип дії над ними [2].

Проблема полягає у неможливості робити асинхронні запити, наприклад на сервер, всередені action-функції. Redux-thunk дозволяє замість дії з цієї функції повертати асинхронну функцію, котра в свою чергу повертає дію. Такий підхід є

необхідним, якщо потрібно робити запити на сервер та відразу малювати зміні на стороні клієнту [3].

Бібліотека Redux-thunk дозволяє дотримуватись найкращих практик під час розробки веб-застосунку, а саме робити запити на сервер не з компоненту, а з action-функції. Ця найкраща практика каже про те, що компонент має лише отримати дані та відмалювати їх, але не має знати звідки ці дані були отримані.

2.2.1.6 Axios

У попередньому підпункті було зазначено про необхідність робити запити на сервер для отримання інформації. Але чим саме робити ці запити? Існує декілька варіантів.

XHR (XMLHttpRequest) – це об’єкт, вбудований в інтерфейс браузера, за допомогою якого можливо робити HTTP запити на сервер. Він є застарілим, і зараз не використовується .

Fetch – новий інструмент, також вбудований в інтерфейс браузера, за допомогою якого можливо робити HTTP запити. Є більш зручним, але має свої недоліки.

Axios – це найкращий інструмент для створення запитів. Їм можливо не тільки робити запити з браузеру, як у випадку попередніх двох інструментів, але й робити запиті з середовища Node.js. Має велику кількість переваг, підтримує async await синтаксис та Promise API. Реалізований з підтримкою Typescript, що є дуже корисним під час розробки додатку з використанням останнього. Має можливість бути повністю налаштованим за потребою розробника, та навіть можливо налаштувати передачу даних для кожного запиту індивідуально [5].

2.2.2 Сервер

Розробка серверної частини відбувається з метою «розвантаження» клієнта. Зберігання даних у БД, читання, оновлення даних, складна обробка інформації та інші операції – всі вони відбуваються на серверній частині веб застосунку.

2.2.2.1 Rest API

Rest API – це один з підходів до архітектури веб-застосунків. Є найчастіше вживаним рішенням. У деяких випадках така архітектура спрощує розробку додатку та надає перевагу у швидкості роботи.

Rest взаємодія між клієнтом та сервером відбувається шляхом надсилання з клієнта HTTP запитів на сервер. Rest архітектура полягає у створені шляхів, так званих «раутів» виду «http://localhost/url» на адрес яких надсилається запит.

Існує чотири основних та найбільш вживаних запитів:

- Get ;
- Post ;
- Put ;
- Delete .

Перший запит - Get використовується лише для отримання даних з сервера, неможливо шляхом цього запиту пересилати інформацію.

Другий запит - Post використовується для надсилання даних на сервер. Типовий приклад – реєстрація нового користувача.

Третій запит - Put використовується для внесення змін в уже існуючі дані. Наприклад, зміна паролю або імені користувача.

Останній запит - Delete використовують для видалення даних, також не має тіла запиту(body).

2.2.2.2 Nest.js

Nest.js – це бібліотека за допомогою якої можлива розробка серверної частини. Розроблена вона на основі іншої back-end бібліотеки – express.js. По факту, Nest.js є своєрідною надбудовою над останньою [6].

Основна перевага цієї бібліотеки – це чітка структура та модульна система. Структура дуже схожа на відомі бібліотеки на яких розробляються великі додатки – Java Spring Boot, C# .NET. Наявність структуру є перевагою через те, що така архітектура є завжди однією і немає необхідності розбиратись, що саме минулий розробник мав на увазі під час написання серверу. Також, схожість з перерахованими бібліотеками надає менший час для входу у розробку.

Модульна система надає дуже легку інтеграцію у нові проекти. Наприклад, вже створений модуль для аутентифікації додати до нового проекту, то після невеликих налаштувань, він буде працювати.

Кожен модуль складається з трьох файлів:

- Сервіс ;
- Контролер ;
- Модуль .

Кожен з цих файлів містить у собі клас з відповідною назвою.

Сервіс – взаємодія з базою даних, запис у неї, читання повний CRUD цикл операцій, тощо.

Контролер – це клас який містить у собі ті самі «раути», іншими словами містить адреси, за якими необхідно звертатися до серверу.

Модуль – це клас який по-суті створений для налаштувань. У ньому збираються до купи сервіс і контролер. Можливі додаткові налаштування, наприклад підключення інших модулів(наприклад модуля з аутентифікацією) або middleware-функції.

2.2.2.3 Typeorm

Будь-якому додатку, якщо він складніший за сайт візитку, необхідно десь зберігати даним. Найпопулярнішим, і напевно, єдиним правильним рішенням є бази даних. Існує два типи баз даних – реляційні та, відповідно, не реляційні. До першого типу відносяться MySQL, SQLite, MariaDB, PostgreSQL.

Реляційні бази є більш зручними для сайтів, де я зв’язок наприклад користувача з товарами. Для створення запитів у реляційні БД було створено мову SQL. Вона є відносно важкою та громіздкою, займає багато місця у коді і у ній не так легко розібратись.

Тому, на заміну SQL, при створенні веб-додатків використовують так звані ORM. **ORM (Object-Relational Mapping)** - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Вони надають API інтерфейс для взаємодії з базою даних. З переваг typeorm над SQL та іншими ORM(Sequilize, тощо) необхідно зазначити:

- Простота ;

- Взаємозаміна з SQL ;
- Інтуїтивно зрозумілість ;
- Займає менше коду .

Також, Nest.js використовує typescript, тому використання typeorm є бажаним у поєднанні з цією back-end бібліотекою.

2.2.2.4 JWT

JSON Web Token (**JWT**) - це відкритий стандарт для створення токенов доступу, заснований на форматі JSON . Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи.

Токен разом з хешуванням паролів за допомогою наприклад bcrypt модуля створюють первинну безпеку для сайту. Токен зберігає у собі необхідну інформацію про користувача і без спеціального ключ-слова декодувати його неможливо. Він використовується для того, щоб лише авторизований користувач, або користувач з додатковими можливостями на сайті, зміг отримати доступ до захищених раутів.

Як приклад, дуже часто у додатку є функціонал доступ до якого має лише авторизований користувач.

2.2.2.5 Passport-jwt

Паспорт у даному контексті означає спосіб у який буде відбуватися аутентифікація користувача. Є дуже багато стратегій:

- Instagram ;
- Facebook ;
- Google ;
- JWT ;
- Twitter ;
- Та багато інших .

JWT стратегія для паспорту є найбільш розповсюдженою для додатків, що використовую свою аутентифікацію та де зареєструватися можливо лише локально, а не за допомогою даних інших соціальних мереж.

Паспорт наразі перевіряє, чи є токен, що прийшов з сервера дійсним.

2.2.3 Висновки до розділу 2

У цьому розділі було проаналізовано та досліджено всі необхідні аспекти для створення веб-додатку. Було проведено порівняння між різними технологіями та знайдені найкращі з них. У розділі зазначено, чому краще за все у контексті цього додатку використовувати саме такий стек технологій.

Було проаналізовано, чому краще використовувати бібліотеку React разом з мовою typescript замість javascript для цього застосунку.

РОЗДІЛ 3. Проектування та розробка додатку

3.1 Архітектура

React підтримує так звану архітектуру SPA(Single Page Application). Тому, саме це архітектурне рішення було закладне в основу створення додатку. Архітектура сайту «на одну сторінку» полягає у тому, що хоча сайт може мати декілька сторінок, наприклад головну та особистий кабінет, по факту він все одно має одну сторінку.

У цій бібліотеці під час створення інтерфейсу для налаштування маршрутизації використовується маленька бібліотека з назвою React-router-dom. Вона надає набір інструментів для реалізації архітектури SPA.

При переходжені за новим покликанням на нову сторінку, фактично тобі малюється новий контент на екрані, без перезавантаження сторінки. Це дозволяє робити менше запитів на сервер та фактично відокремити клієнт і сервер.

При подальшій розробці додатку, були прийняті наступні архітектурні рішення:

- Використання Typescript ;
- Розподілення компонентів за типами ;
- Написання максимально універсальних компонентів ;
- Використання Redux ;
- Використання підходу створення власних хуків ;
- Винесення логіки в окремі файли .

Використання Typescript. Цю надбудову над основною мовою веб-розробки було використано з метою позбавлення всіх недоліків динамічної та

слабкої типізації. Також, використовувати typescript під час розробки є необхідним для клієнта, адже серверна частина та взаємодія з базою даних реалізовані саме на ньому.

Розподілення компонентів за типами. Під час написання додатків та створення компонентів, останні поділяють на два умовні типи – так звані «дурні» та «розумні». Перший тип лише приймає дані і малює те що необхідно на екран, не проводячи жодних маніпуляцій за даними. Другий тип утримує всередині бізнес-логіку, як наприклад модальні вікна або будь які інші хуки, сервіси тощо.

Написання максимально універсальних компонентів. Все що було написано в коді має бути описано там лише один раз. Іншими словами, замість того щоб писати два чи три вікна для вводу даних, краще написати один з можливістю максимального розширення. При такому підході розробки, спочатку створюється максимальна кількість «маленьких компонентів» з яких потім збираються сторінки та інші компоненти.

Використання Redux. Цю бібліотеку було використано для спрощення обміну даними у всьому додатку. Також, як було зазначено в теоретичній частині, компонент не має знати звідки він отримує ці дані. Тому Redux разом з Redux-thunk ідеально підходять для реалізації цього підходу.

Створення власних хуків. Після того як у бібліотеці React відмовилися від використання класових компонентів та перейшли на функціональні, для збереження, завантаження та взаємодії використовують спеціальні функції – так звані хуки(hooks).

Їх перелік не дуже великий:

- useState ;
- useEffect ;
- useRef ;

- useCallback ;
- useMemo .

Це перелік основних та частіше за все вживаних хуків. Але, гарним підходом під час розробки є створення власних. Мною був написаний наступний функціонал – useDimension (для роботи з розмірами екрану), useDispatchFunc (використовується для роботу з action-функціями, що є необхідними для роботи з redux), useFormHandler (для обробки форм введення, наприклад використовується під час додаванні нової книги), usePreload (потрібен для завантаження даних з сервера під час завантаження сторінки), useStateParams (використовується для пошука користувача або книг), useVisibility (потрібен для взаємодії з модальними вікнами), useSearch (специфічний хук, потрібен для реалізації пошукової системи).

Винесення логіки в окремі файли. Задля того щоб не навантажувати компонент та зробити його якнайменш громіздким, деякі функції виносяться в окремі файли.

3.2 База даних

3.2.1 PostgreSQL

PostgreSQL - вільна об'єктно-реляційна система управління базами даних (СКБД).
Має наступні обмеженні:

Ця БД була обрана через такі фактори як: зручність використання, повна сумісність з обраною ORM та те, що база є реляційною. У подальшій схемі будуть позначені використані відношення між таблицями, котрі було б складно реалізувати у випадку з нереляційними базами даних.

3.2.2 Структура бази даних

База даних була побудова за наступною ER моделлю. ER модель – це модель даних, що використовується під час проектування архітектури бази даних.

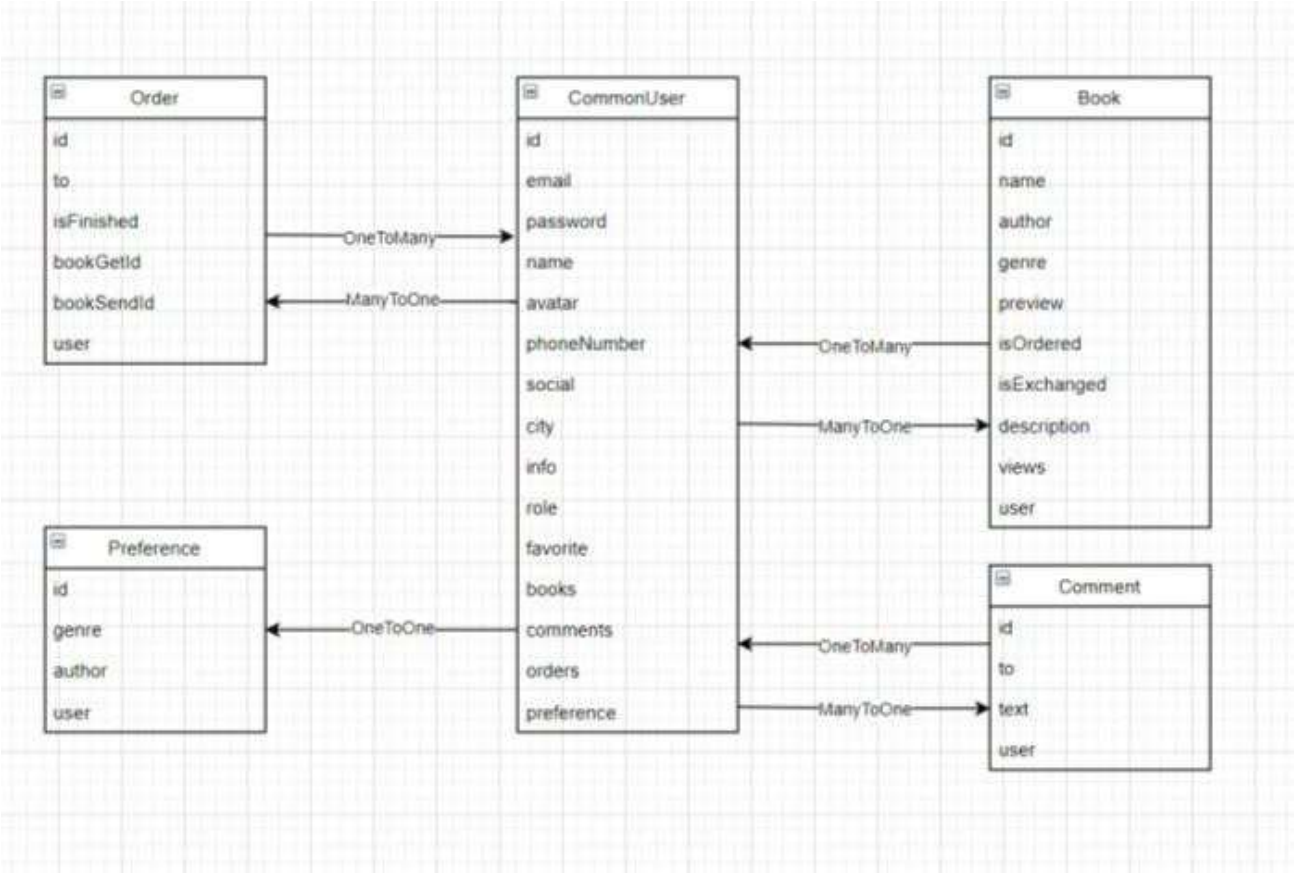

Рис 1.1. ER модель бази даних

Як зазначено у схемі, всі таблиці по’язані з користувачем. Один користувач може мати унікальні книжки, які мають відношення один до багатьох та багато до одного. Кожен користувач має свої коментарі, історію замовлень та збережені дані про рекомендації.

Більшість з таблиць з таблицею користувача мають відношення один до багатьох. Це зроблено з щоб описати таку схему – один користувач може мати багато книг, коментарів, замовлень – але кожне з перерахованих належить лише одному користувачу.

Таблиця, для надання рекомендацій має відношення один до одного, адже одному користувачу належать лише одні рекомендації.

Для кожного відношення у базі даних додатку є налаштований параметр CASCADE, з метою якщо користувача буде видалено, то і всі його відношення будуть видалені також.

Typeorm надає більш просте рішення для того, аби не прописувати міграції БД. При ініціалізації підключення до БД (наприклад, під час перезавантаження сервера) всі зміни в Entity будуть внесені і в таблиці в базі даних.

3.3 Алгоритм надання рекомендацій

Як було зазначено в теоретичній частині, розробка алгоритму надання рекомендацій – дуже складний процес, що потребує необхідних знань. Для даного застосунку, я розробив узагальнений алгоритм для надання рекомендацій.

Для кожного користувача при реєстрації створюється індивідуальна таблиця, у якій зберігаються зібрані дані про тих авторів та жанри які йому сподобались. Інформація зберігається неявним шляхом – під час додавання в улюблене, перехід на сторінку книги, тощо. Також у базі даних зберігаються інтервали запитів. Це необхідно задля відсікання неактуальної інформації.

Ще одна ймовірна ситуація – це ситуація, коли користувач нещодавно зареєструвався у додатку. Зрозуміло, що ніяких даних про нього зібрано ще не було. Для цього використовується принцип середнього згладження – коли при

створенні рекомендації використовують не лише дані про користувача, але і усереднені дані всіх користувачів.

У моєму випадку я беру половину даних всіх користувачів та об’єдную з даними для конкретного користувача. У такому разі, навіть новий користувач буде мати неперсоналізовані рекомендації.

3.4 Розробка застосунку

Розробку застосунку можна поділити на дві частини – розробка front-end та розробка back-end.

3.4.1 Розробка клієнта

Клієнтська частина була створена у стеку React+typescript / redux(thunk)/ axios.

Додаток має чотири сторінки:

- Головна ;
- Книги ;
- Кабінет ;
- Адміністратор .

Неавторизований користувач може лише переглянути правила, зареєструватись та переглянути всі книги, але не має можливості робити жодні маніпуляції окрім перерахованих. Всі посилання захищено – якщо навіть користувач знає адресу наприклад особистого кабінету, відкрити він його не зможе, система направить його на головну з метою проходження аутентифікації.

Головна. Під час завантаження додатку ми отримуємо сторінку, де є можливість переглянути правила порталу, зареєструватися та увійти в систему. Після того як дані були введені та користувач пройшов аутентифікацію, йому стають доступним нові посилання на панелі керування зверху.

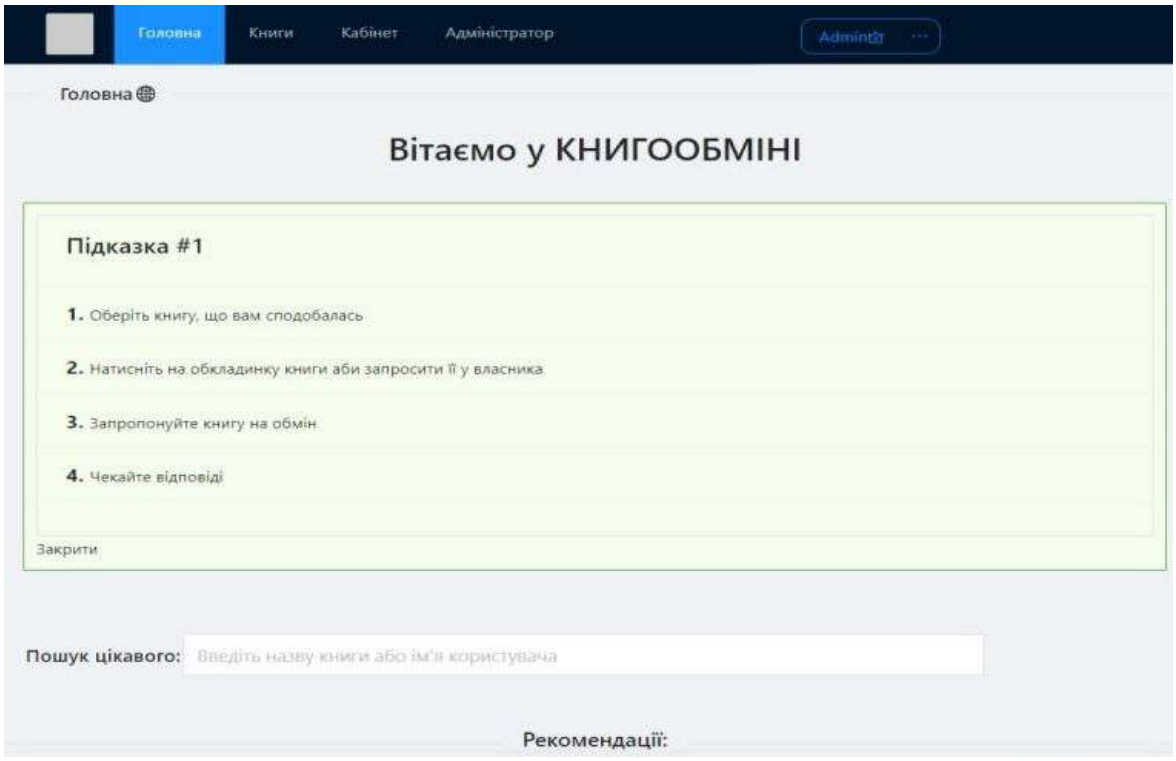

Рис 2.1 – Головна сторінка

На цій сторінці маємо підказку, яку можливо закрити аби вона не заважала, котра описує «життєвий цикл» обміну книжками.

Маємо пошук, що реалізовано з автопідказками – якщо людина введе частину слова, що міститься в імені користувача, назві або авторі книги, то з’явиться панель з можливими варіантами.

Якщо поле пошуку є порожнім, то користувачу будуть надані підібрані рекомендації.

Книги. Ця сторінка має основну мету – показ всіх книг, що були додані в систему. Також, на ній можливо додати нову книгу шляхом взаємодії з спеціальною кнопкою.

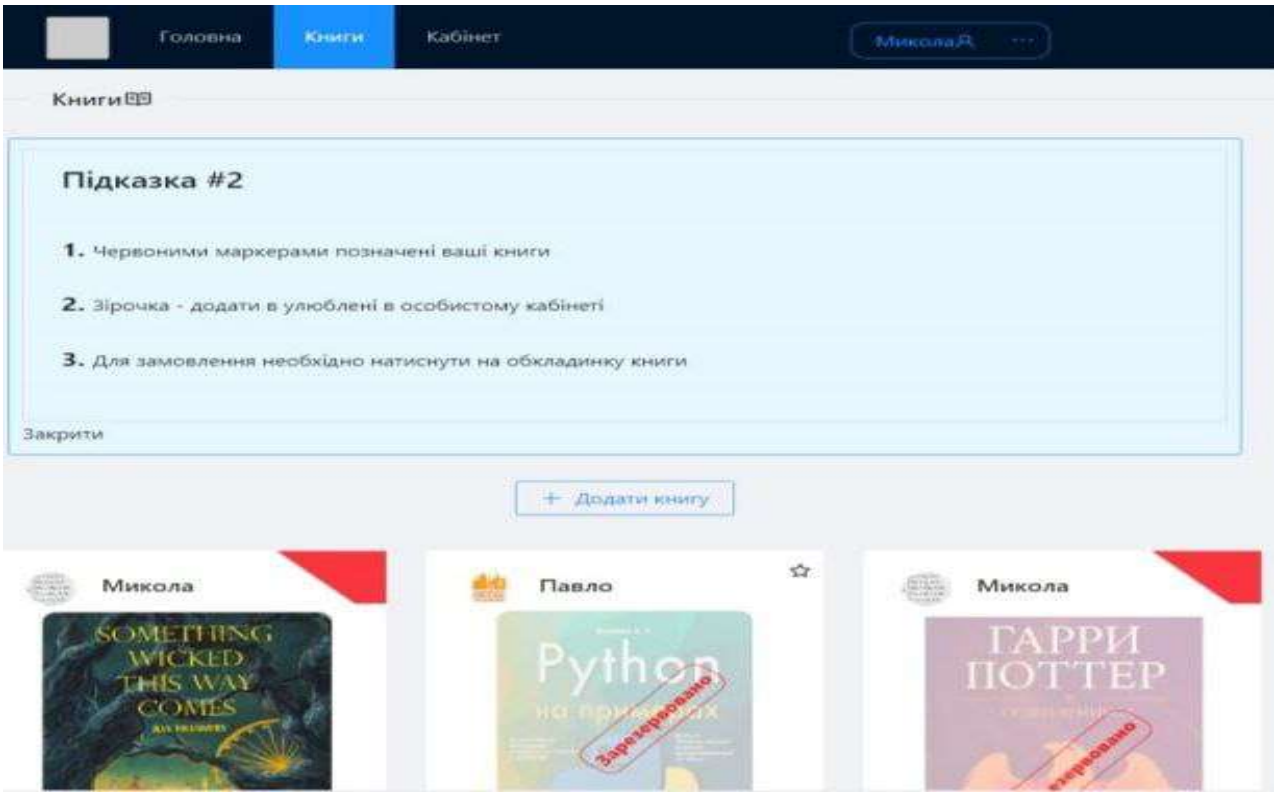

Рис 2.2 – Сторінка книг

При відкриванні сторінки автоматично завантажуються дані про всі книги. Зверху, як і на головній, є підказка, що розповідає про неочевидні елементи інтерфейсу. Наприклад – червона позначка, означає, що книга моя.

Всі книги представлені у вигляді карток, де розташована фотографія обкладинки автор жанр та якому користувачу належить. Якщо натиснути на аватар користувача, відкриється сторінка користувача де можливо почитати дані про нього – як зв’язатись, інформація про улюблені жанри літератури, всі книжки які він додав. Також є можливість додати відгук на користувача.

Знизу книги є посилання на сторінку книги, де є можливість почитати опис.

Якщо натиснути на фотографію на карточці книги то ми зможемо додати нове замовлення, що буду збережено в особистому кабінеті. При натисканні з’явиться модальне вікно у якому можна вибрати книгу, щоб запропонувати на обмін.

Коли відкривається обмін обидві книги (та, що хочуть отримати і та що пропонують) отримують статус зарезервовано. У такому разі інші користувачі не можуть замовити її.

Особистий кабінет. У особистому кабінеті зазначена інформація про користувача – фото, додаткова інформація, телефон. Також є 4 «таби». Таби – своєрідні вкладки, які утримують у собі певний контент. В особистому кабінеті їх чотири:

- Отримання ;
- Відправлення ;
- Улюблені ;
- Мої книги .

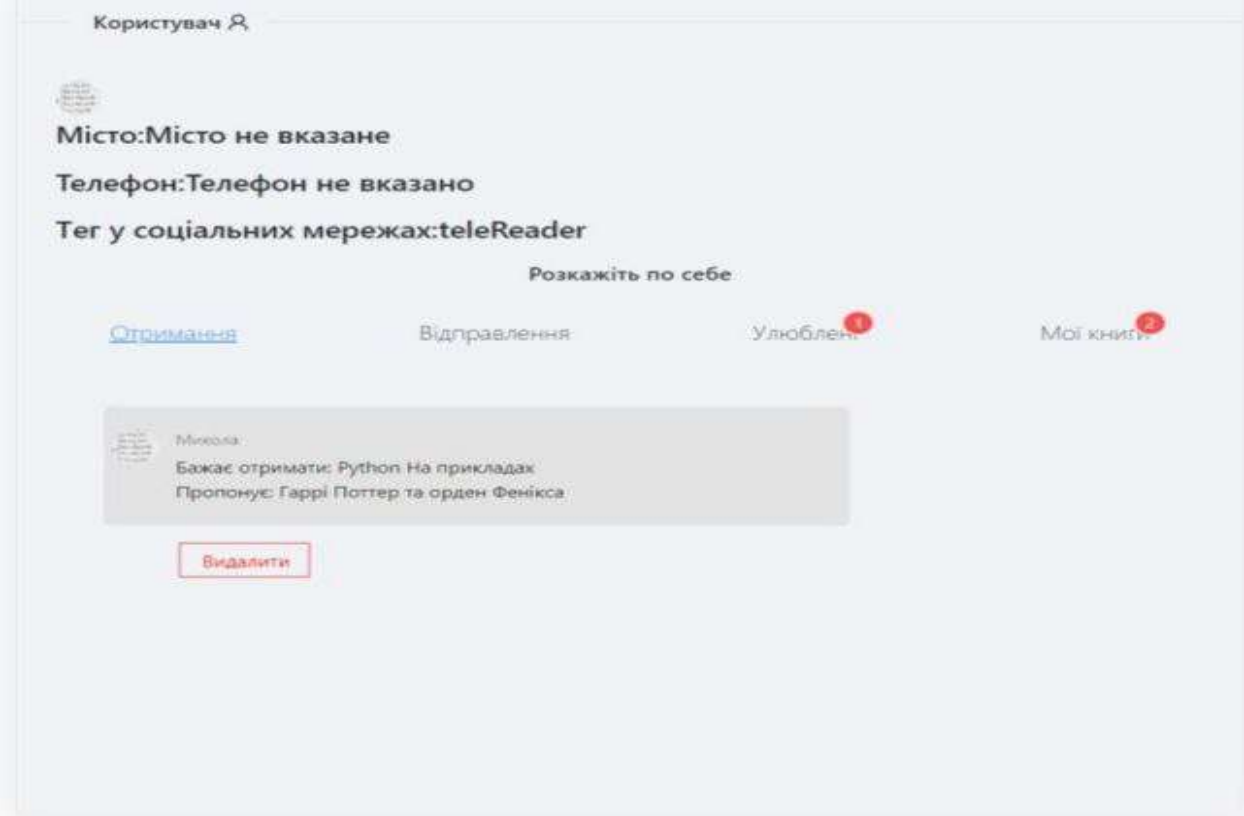

Рис 2.3 – Особистий кабінет

Отримання – це всі замовлення, що зробив. Їх я можу прибрати(видалити) якщо я зробив його помилково або за бажанням.

Відправлення – це всі замовлення, щ надіслали мені. Я можу або відхилити замовлення, або прийняти. У разі відхилення, замовлення просто буде видалено. Якщо ж замовлення було прийняте, то картка з інформацією про замовлення переходить в історію, а обидві книги не видаляються з бд, але є більше недоступними для замовлення або перегляду.

Улюблені – це ті книги, що користувач додав для подальшого переглядую

Мої книги – це ті книги, що додавав я. Їх користувач має можливість редагувати за своїм бажанням, або видалити.

Адміністратор. Ця сторінка доступна лише тим користувач, що мають роль адміністратора в системі. Їх основна мета – видалення недоречного контенту або користувачів з системи, що порушили правила.

Має також три таби:

- Всі книги ;
- Всі користувачі ;
- Історія замовлень .

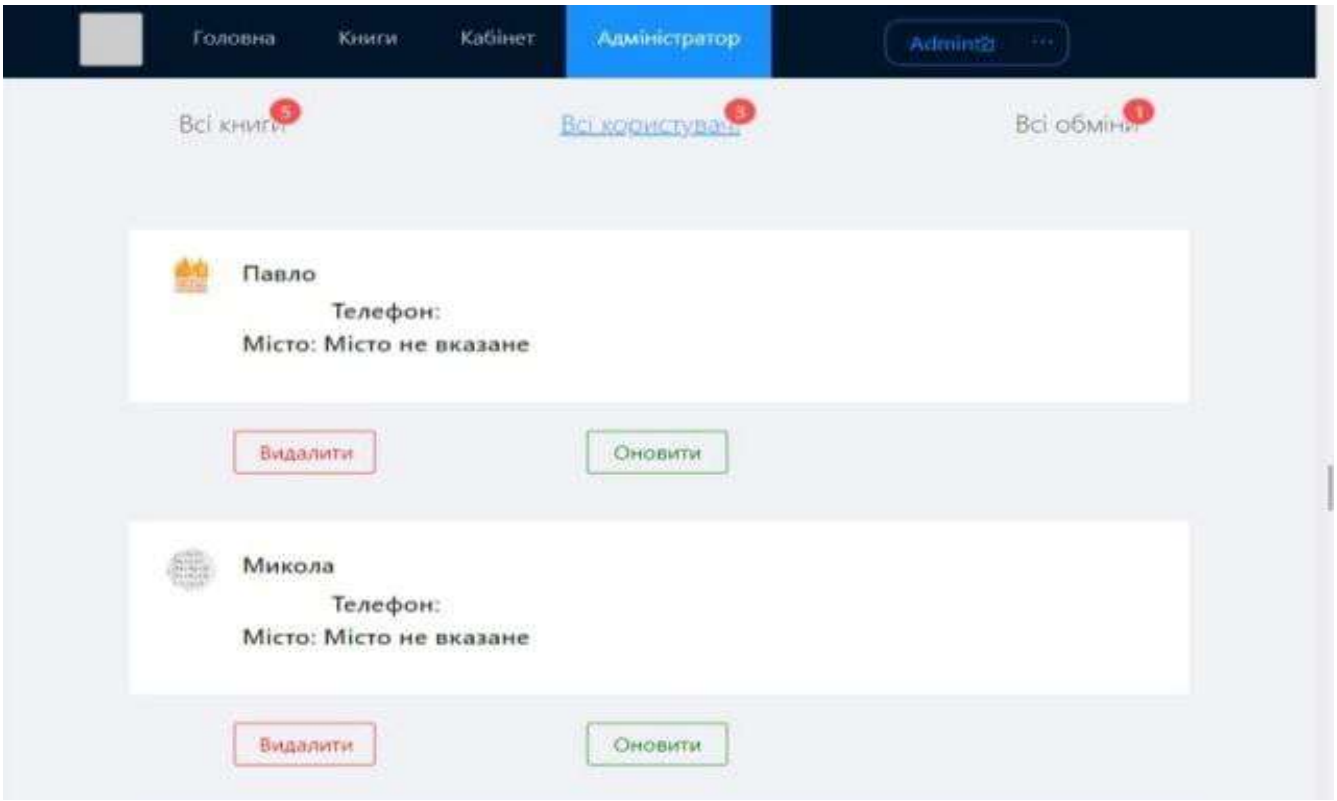

Рис 2.4 – Сторінка адміністратора

Якщо назва перших двох повністю описує функціонал(видалення книг з системи, видалення користувачів або зміна ролі користувача), то остання потрібна для перегляду історії замовлень та вирішення конфліктних ситуації.

3.4.2 Розробка сервера

Для розробки серверної частини було обрано Rest архітектуру на основі якої і було створено веб-додаток.

Перед стартом сервера, перш за все typeorm намагається підключитись до бази даних. Якщо до бази під'єднатися неможливо і виникла помилка, то помилку буде записано у консоль, а сервер не буде запущено. Це зроблено з метою уникнення таких проблем, як наприклад завантаження додатку, але неможливості увійти в систему через те що до БД неможливо під'єднатися.

У пункті «Структура бази даних» було описано всі таблиці та взаємодії між ними. Оскільки Nest.js підтримує модульну систему, для взаємодії з кожною таблицею було створено окремий модуль – для запису, читання, видалення, тощо.

Тільки для таблиці замовлень не було створено окремого модуля через те, що він дуже тісно пов'язаний з користувачем і їх можливо об'єднати.

Також було створено окремий модуль, так званий Shared-module. Він створений для аутентифікації та авторизації користувача. Задля того, щоб не писати кожного разу одне й те саме, всю логіку пов'язану з входом, реєстрацією та захистом раутів винесено в окремий модуль.

Модуль для адміністрації, хоча по факту і взаємодіє з книгами та користувача, біло винесено в окрему директорію. Це зроблена через те, що доступ до певних функцій може отримати лише адміністратор. Тому, всі раути захищені через middleware-функцію на перевірку ролі. Така функція має доступ до об'єктів request та response і перед тим як передати дані у сервіс вона може робити з ними певні маніпуляції. У даному випадку, якщо токен та користувач, що у ньому зашифрований, не має ролі адміністратора, то функція не дозволить робити подальших маніпуляцій.

Задля захисту додатку було обрано найпоширенішу стратегію – захист даних через json web token та хешування паролів за допомогою модуля bcrypt.

3.5 Висновки до розділу 3

У розділі було описано архітектуру веб-додатку. Було прийнято рішення побудови front-end та back-end.

За основу було взято SPA підхід для розробки клієнта. Також, використовувалися найкращі та найпоширеніші практики, а саме – універсальні компоненти, hook as a service підхід.

Описано процес розробки клієнту та захисту клієнту від небажаного втручання. Кожна сторінка має, що індивідуальний функціонал, також була описана з точки зору процесу розробки.

Під час розробки серверної частини було використано модульний підхід, за допомогою обраного фреймворку. Захист додатку біло зроблено через jwt та модуль bcrypt.

Було описано алгоритм надання рекомендації, та процес аналізу алгоритму.

Також, зазначено та аргументовано основні інтерфейсні рішення.

ВИСНОВОК

Мета моєї курсової роботи – вирішення проблем обміну паперовою літературою. Задля цього було створено веб-застосунок, що спрощує цей самий обмін.

Під час дослідження теоретичної частини, було обрано найпопулярніші практики та інструменти для розробки додатку. При розробці застосунку був доданий адаптивний дизайн, адже нерідко люди користуються застосунком з телефону.

Саме через обрані інструменти розробки додаток є може бути легко розширено згодом, доданий новий функціонал, або навіть перехід додатку до нової тематики – обмін не лише книжками а чим завгодно.

Побудований алгоритм надання рекомендації, хоча і не може замінити більш потужні аналоги, але надає правильні рекомендації в умовах розробленого застосунку.

Говорячи про подальший розвиток додатку, можливо створити гарантійну систему(страхування вартості) книги – при оформлені обміну, користувач оцінює вартість книги і у разі проблем під час обміну буде компенсована її вартість.

Підводячи підсумок, можливо зазначити що розроблений додаток, через практичну відсутність аналогів, може бути широковживаних не лише в НаУКМА, але навіть поза її межами.

Список джерел

1.Документація React[Електронний ресурс] : https://ru.reactjs.org/

2.Документація Redux[Електронний ресурс] : https://redux.js.org/

3.Документація Redux-thunk[Електронний ресурс] :
https://github.com/reduxjs/redux-thunk

4.Документація Typescript[Електронний ресурс] : https://www.typescriptlang.org/

5.Документація Axios[Електронний ресурс] :
https://www.npmjs.com/package/axios

6.Документація Nest.js[Електронний ресурс] : https://docs.nestjs.com/

7.Документація Typeorm[Електронний ресурс] : https://typeorm.io/#/

8. Документація Passport-jwt[Електронний ресурс] :
http://www.passportjs.org/packages/passport-jwt/

9.Анатомия рекомендательних систем. Часть первая[Електронний ресурс] :
https://habr.com/ru/company/lanit/blog/420499/

10. Документація Antd[Електронний ресурс]: https://ant.design/