МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «Web-технології»

на тему:

**«**Розробка CMS системи інтернет-магазину книг**»**

студента II курсу групи ІПЗ-20-4

спеціальності 121 «Інженерія програмного забезпечення»

Кормиша Романа Івановича

(прізвище, ім’я та по-батькові)

Керівник: Чижмотря О.Г .

Дата захисту: " \_\_\_ " \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

(підпис) (прізвище та ініціали) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

(підпис) (прізвище та ініціали)

Житомир – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Факультет інформаційно-комп’ютерних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень: бакалавр

Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»

В.о зав кафедри ІПЗ

\_\_\_\_\_\_\_\_\_\_\_\_ А.В Морозов

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2021 р.

ЗАВДАННЯ

НА КУРСОВИЙ ПРОЕКТ СТУДЕНТУ

Кормишу Роману Івановичу

1. Тема роботи: Розробка інтернет-магазину книг

керівник роботи: старший викладач кафедри ІПЗ, Чижмотря Олена Генадіївна .

1. Строк подання студентом: “ ” 2021р.
2. Вихідні дані до роботи: Розробити інтернет-магазин книг – розробити та реалізувати систему адміністрування для сайту
3. Зміст розрахунково-пояснювальної записки (перелік питань, що підлягають розробці)
   * + 1. Постановка завдання
       2. Аналіз аналогічних розробок
       3. Алгоритми роботи програми

4. Опис роботи програми

5. Програмне дослідження

1. Перелік графічного матеріалу(з точним зазначенням обов’язкових креслень)

1. Презентація до КР\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Посилання на репозиторій: https://gitlab.com/2020-2024/ipz-20-4/kormysh-roman/course-work-2.git

1. Консультанти розділів проекту (роботи)

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посади консультанта | Підпис, дата | |
| завдання  видав | завдання прийняв |
| 1,2 | Чижмотря О.Г., ст. викладач каф. ІПЗ |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата видачі завдання “ ” 2021 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Примітки |
| 1 | Постановка задачі | 4.12.21 – 5.12.21 | Виконано |
| 2 | Пошук, огляд та аналіз аналогічних розробок | 5.12.21 – 6.12.21 | Виконано |
| 3 | Формулювання технічного завдання | 6.12.21 – 8.12.21 | Виконано |
| 4 | Опрацювання літературних джерел | 8.12.21 – 12.12.21 | Виконано |
| 5 | Проектування структури | 12.12.21 – 14.12.21 | Виконано |
| 6 | Написання програмного коду | 14.12.21 – 31.12.21 | Виконано |
| 7 | Відлагодження | 01.01.22 –  10.01.22 | Виконано |
| 8 | Написання пояснювальної записки | 05.01.22 –  10.01.22 | Виконано |
| 9 | Захист | 13.01.22 |  |

**Студент***\_\_\_\_\_\_\_\_\_\_\_\_\_\_* Кормиш Р.І

(підпис) (прізвище та ініціали)

**Керівник проекту** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чижмотря О.Г

(підпис) (прізвище та ініціали)

**РЕФЕРАТ**

Пояснювальна записка до курсового проекта на тему «Розробка інтернет магазину книг» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 32 сторінках друкованого тексту.

Пояснювальна записка має 101 сторінки додатків. Список використаних джерел містить 12 найменувань і займає 1 сторінку. В роботі наведено 23 рисунів. Загальний обсяг роботи – 133 сторінок.

У першому розділі було обґрунтовано створення сайту на тему Розробка інтернет-магазину книг.

У другому розділі проведено проектування і розробка сайту.

У третьому розділі проведено тестування програмного продукту.

Висновок містить в собі результати виконаної роботи при створенні сайту на тему «Розробка інтернет-магазину книг».

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова:

MYSQL, JAVASCRIPT, PYTHON, FAST API, VUE JS, PET-ПРОЕКТ, WEB-САЙТ, MVC, СЕРВЕР, АДМІН, АВТОРИЗАЦІЯ, МАГАЗИН, ІНТЕРНЕТ, ДАНІ.

# 

**Зміст**

[**ВСТУП** 7](#_Toc92540836)

[РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ 8](#_Toc92540837)

[**1.1** **Аналіз задачі, засобів та методів її вирішення** 8](#_Toc92540838)

[**1.2** **Аналіз існуючого програмного забезпечення за тематикою курсової роботи.** 8](#_Toc92540839)

[**Висновки до першого розділу :** 15](#_Toc92540840)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16](#_Toc92540841)

[**2.1 Проектування загального алгоритму роботи веб-додатку** 16](#_Toc92540842)

[Загальна схема роботи програми : 17](#_Toc92540843)

[**2.2 Розробка функціональних алгоритмів роботи програми** 18](#_Toc92540844)

[**2.3 Розробка програмного забезпечення** 19](#_Toc92540845)

[РОЗДІЛ 3 ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ 24](#_Toc92540846)

[**3.1 Опис роботи з програмним додатком (Опис інтерфейсу)** 24](#_Toc92540847)

[**3.2** **Тестування роботи програмного забезпечення** 30](#_Toc92540848)

[**Висновки до 3 розділу :** 31](#_Toc92540849)

[**ВИСНОВКИ** 32](#_Toc92540850)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 33](#_Toc92540851)

[**ДОДАТКИ** 34](#_Toc92540852)

**ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ**

БД – База даних

КП – курсовий проект

MVC – Модель–вигляд–контролер (або Модель–представлення–контролер, Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

ПЗ – Програмне забезпечення

# **ВСТУП**

У цій курсовій роботі буде наведено процес розробки інтернет-магазину книг.

Інтернет-магазин книг – це інтернет-магазин, де корситувач може замовляти книги. Взагалом останнім часом інтернет-магазини ствють все більш і більш популярними, тому що дозволяють здійснювати покупки не виходячи з дому, тому дана тема курсової є актуальною.

**Метою створення** даного курсового проекту є розробка CMS системи інтеренет магазину книжок.

**Об’єктом дослідження** є методи та засоби розробки CMS за визначеними предметними областями.

**Предмет дослідження** є використання веб-технологій для забезпечення інформаційних потреб предметної області.

# РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

## **Аналіз задачі, засобів та методів її вирішення**

Завдання полягає в тому, що потрібно:

1. Визначити потреби звичайного користувача
2. Зрозуміти основні принципи та алгротитми роботи інтернет-магазину
3. Продумати можливість додавання книжок та їх повного редагування
4. Реалізувати сортування та пошук книг
5. Реалізувати userfriendly інтерфейс
6. Продумати розподілення ролей та можливість створення нових адмінів
7. Створити корректну базу даних
8. Продумати дизайн сайту
9. Реалізувати адаптивність для різних пристроїв

Для пришвидчення роботи інтренет-магазину потрібно використовувати асинхронні методи.

## **Аналіз існуючого програмного забезпечення за тематикою курсової роботи.**

При аналізі вже існуючих *веб-сайтів* за тематикою курсової роботи було виявлено декілька проектів. Всі вони схожі і мають подібну реалізацію, адже всі вони є інтернет магазинами книжок, які призначені для продажу книжок.

Для розробки сайту використано мови програмуваня Python(+фраймворк Fast API) та JS(+фреймворк Vue js)

Aналогічні інтренет-магазини за заданою темою:

**Yakaboo**

Посилання : https://www.yakaboo.ua/

Розглянемо даний аналог:

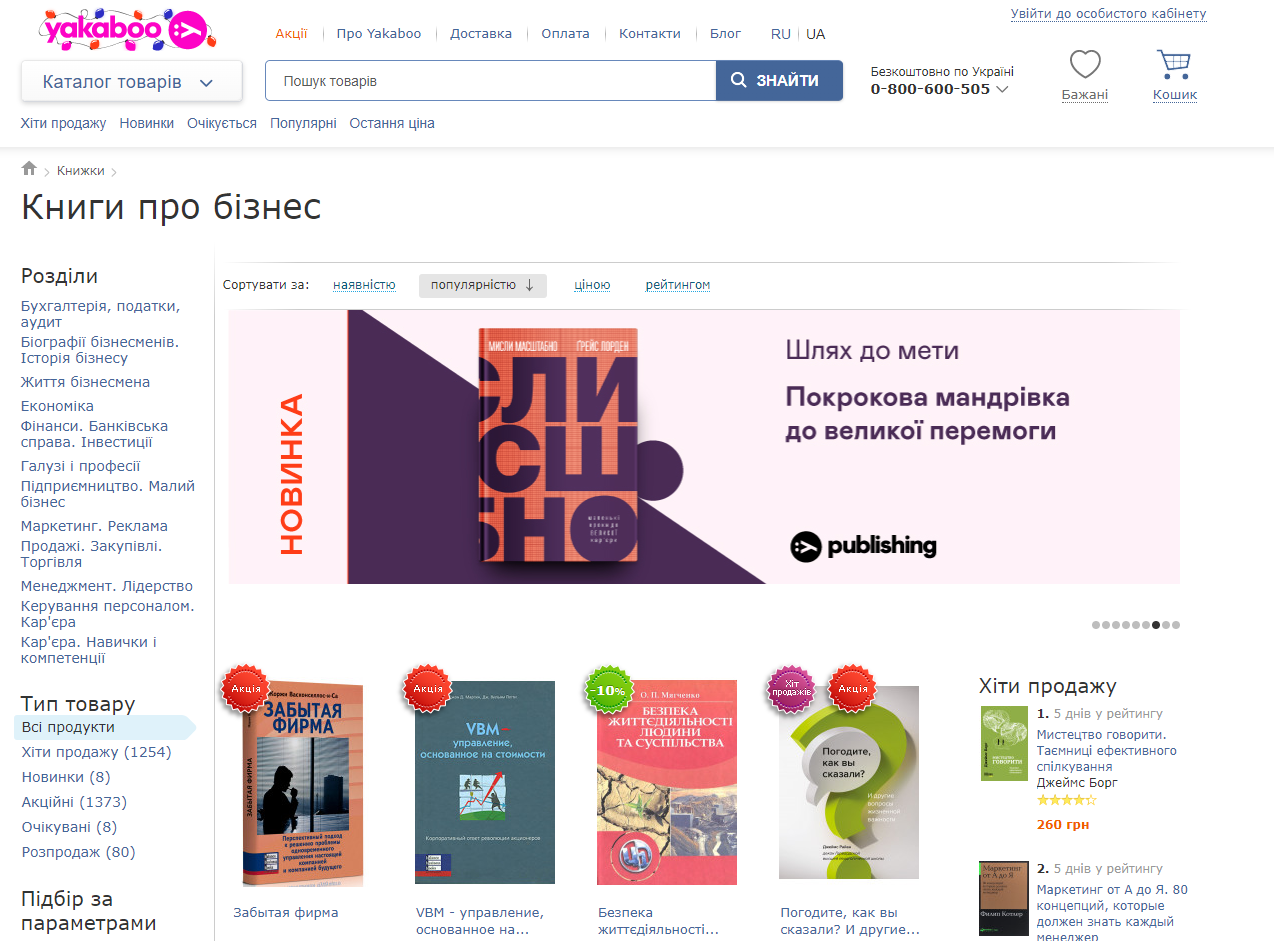


Рис 1.1 Yakaboo

Інтернет-магазин Yakaboo є одним із найпопулярніших інтернет-магазинів про продажу книг, адже має інтуїтивно зрозумілий користувацький інтріейс та широкий асортимент книг. Також має широкий вибір параметрів для сортування товарів. Серед мінусім можна виділити відсутність адаптиву.

Основні плюси:

* Інтуїтивно зрозумілий користувацький інтерфейс
* Широкий вибір параметрів для сортування товарів
* Широкий асортимент книг

Основні мінуси:

* Відсутність адаптиву

**Amazon**

Посилання : https://www.amazon.com/

Розглянемо даний аналог:

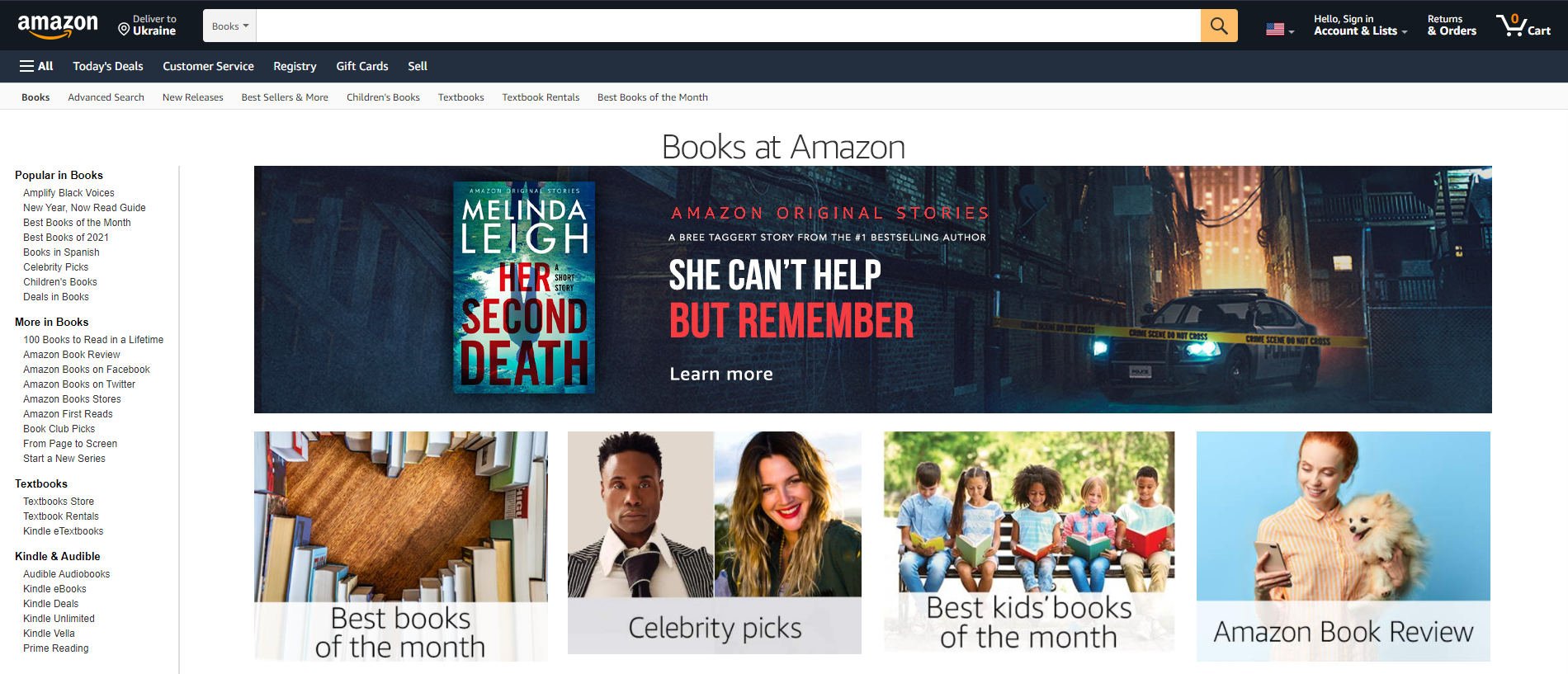


Рис 1.2 Amazon

Інтернет-магазин Amazon є одним із найперших інтернет-магазинів.

Спочатку даний сайт розроблявся саме для продажу книг, а вже згодом розширювавля для інших видів товарів. Також має широкий вибір параметрів для сортування товарів. Серед мінусім можна виділити відсутність адаптиву та незначно надмірну уількість елементів користувацького інтерфесу.

Основні плюси:

* Широкий вибір параметрів для сортування товарів
* Широкий асортимент книг
* Один з найперших інтернет-магазин книг

Основні мінуси:

* Відсутність адаптиву
* Незначно надмірний користувацький інтерфейс

**British Book**

Посилання: https://www.britishbook.ua/

Розглянемо даний аналог:



Рис 1.3 British Book

Інтернет-магазин British Book це інтернет-магаизин, який має стильний дизайн та можливості попередніх розглянутих продуктів. Навідміну від попередньо-розглянутих сайтів, має також адаптив.

Основні плюси:

* Всі, що і в попередньо-розглянутих інтерент-магазинів
* Адаптив для різних пристроїв

Основні мінуси:

* Не такий відомий як попередньо-розглянуті магазини

Актуальним напрямком реалізації власного інтернет-магазину є створення веб-сайту з бізнес логікою подібною до реалізованої у всіх представлених продуктах, реалізувати користувацький інтерфейс по принципам yakaboo, уникаючи надмірності Amazon, але послідувати основним аспектам інтерфейсу цього продукту. Також продумати сучасний дизай сайту, звертаючи увагу до дизайну British Book та реалізувати адаптив.

## **1.3 Технічне завдання на курсову роботу**

**1. Загальне положення**

***1.1Найменування програмного засобу***

Повне найменування програмної системи: "Розробка інтернет-магазину книг" (надалі "веб-додаток"). Коротка назва програмної системи - " Інтернет-магазин"

***1.2. Призначення розробки та область застосування***

Веб-сайт "Інтернет-магазин книг" призначений для розміщення каталогу книг, їх пошуку, та продажу. Система може бути впроваджена у роботу інтернет-магазину. Веб-сайт " Інтернет-магазин книг" дозволить швидко переглядати інформацію про книги, та забезпечить керівництву підприємства своєчасною та повною інформацією про актуальний стан замовлення покупок інтернет-магазину у повному обсязі, а також надасть користувачам можливість вибору книг різних видавництв в залежності від їх смаку.

***1.3. Найменування розробника та замовника.***

Розробник даного продукту - студент групи ІПЗ-20-4 Кормиш Роман Іванович (надалі "розробник"). Замовник програмного продукту – кафедра інженерія програмного забезпечення Житомирського державного технологічного університету в межах виконання курсової з дисципліни «Web-тееології» Чижмотря Олена Геннадіївна, Морозов Андрій Васильович **2. Підстава для розробки**

***2.1. Документ на підставі якого ведеться розробка***

Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія програмного забезпечення».

**3. Вимоги до програми**

***3.1. Вимоги до функціональних характеристик.***

***3.1.1. Загальні вимоги***

Веб-додаток має забезпечувати:

• можливість дистанційної роботи з робочих станцій локальної та глобальної мережі підприємства;

• постійний доступ користувачів веб-додатку;

• оформлення замовлення;

• організацію управління сайтом;

• можливість доступ до бази даних;

***3.1.2. Склад виконуваних функцій***

Розробити інтернет-магазин книг, що підтримує виконання наступних операцій:

1. Пошук книг.
2. Зміна інформації про книгу.
3. Додавання запису про книгу:

* Назва
* Дата видання
* Ціна
* Автор
* ISBN
* Жанри
* Короткий опис

1. Фільтрація книг за параметрами:

* Назва
* Автор
* Дата видання
* Кількість
* Ціна

1. Перегляд інформації про книги

***3.1.3. Організація вхідних і вихідних даних***

Вхідними даними є інформація про книгу (Назва, Автор, ціна, кількість, дата видання, ISBN, опис)

***3.1.4. Часові характеристики і розмір пам'яті, необхідної для роботи програми.***

* Час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 0,25 с.
* Час виконання команд меню не більше 1 с.
* Відображення масивів даних за запитами не більше 3 хвилин.
* Доступність БД – 90% цілодобово.
* Операції з’єднання з БД не більше 1 хвилини.
* Обсяг оперативної пам'яті, необхідний для роботи програми не менше 1Гб.
* Дисковbй простір, необхідний для збереження програми і файлів даних не більше 300 Мбат для робочої станції та 20 ГБайт.
* Інсталяційний пакет програми, що містить у складі БД не повинні перевищувати 100 Мбайт.

Схема системи інтернет-магазину книг:

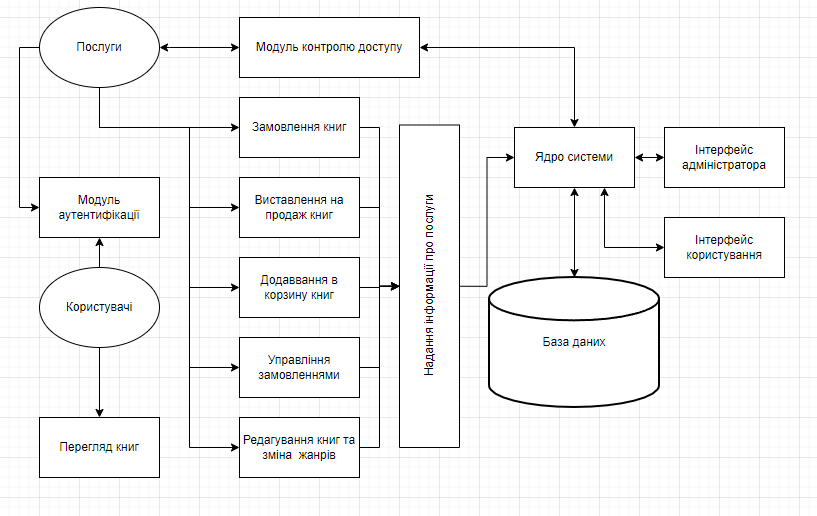
****

Рис 1.4 Структура системи інтернет-магазину книг

## **Висновки до першого розділу :**

У ході виконання першого розділу було поставлено завдання проекту, а саме який має бути функціонал і визначено актуальність проекту. Також визначено мови і фреймворки для розробки та актуальний напрямок розробки власного продукту. Розглянуто існуючі програмні продукти зі схожою тематикою та більш детально проаналізовано деякі із них, а саме : Amazon, British Book та Yakaboo.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## **2.1 Проектування загального алгоритму роботи веб-додатку**

На даному етапі розробки розглянемо загальний алгоритм роботи інтернет-магазину. Загальна схема роботи приведена на рисунку 2.1.

Для початку роботи потрібно створити моделі бази даних. Потум спроектувати схеми з унаслідуванням, адже саме це потрібно для API, який буле реалізовуватися на мові програмування Python та його фреймворку Fast API.

Зголом потрібно реалізувати основні роути для виконання запитів, що дозволить протестувати отримання та обробку інформації отриманої від даного API. Потім реалівувати всі роути та сервіси для кожної сутності, з використання схем які ми описали раніше.

Для авторизації використаємо JWtoken, для забезпечення безпеки даних – хешування паролів.

Тепер спроектуємо візуальну частину. Для цього використаємо мову програмування JS та її фреймворк Vue js. Напишемо декілька компонентів та першу сторінку, з використанням vue-store, що протестувати з’єднання зі створеною API. Потім потрібно написати всі шаблони сторінок, стилі та інші компоненти з сторінками.

Для взаємодії з користувачем потрібно реалізувати “діалогові вікна” (форми). Для того, щоб гість, зареестрований користувач і адмін мали різні права, реалізуємо розділення на ролі з різними прававми. Де адмін має найбільші права (право створювати адмінів включно), користувач – набір повноцінних прав, а гісті може тільки переглядати деякі сторінки сайту.

При запуску сайту користувач буде бачити основну сторінуку, сторінку магазину. Для йому буду пропонуватися зареєструватися або аутентифікуватися. Після цих дій він отримає можливість замовляти книги та виставляти на продаж власні.

## Загальна схема роботи програми :

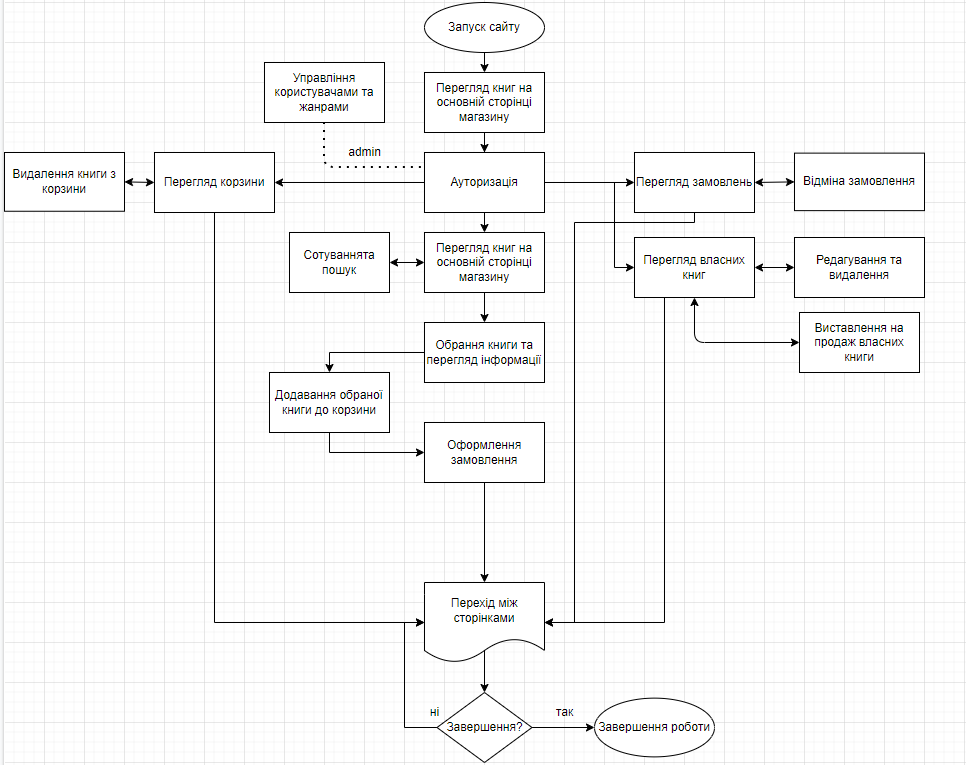


Рисунок 2.1 – Загальна схема роботи програми

## **2.2 Розробка функціональних алгоритмів роботи програми**

Залишилося розглянути детальніше алгоритми робити основних функцій.

Звичайно, основне призначення інтренет-магазину книг – продаж книг, тому розглянемо саме це. Для висталвення книги на продаж потрібно заповнити відповідну форму, після чого книга розміститься на сайті. При заповненні форми будуть з’являтися підзказки про коректність вводу. Після розміщення інший користувач зможе замовити цю книги в максимальній кількості вказаній продавцем. Користувач додає книгу в корзину, звідки можна оформити замовлення. Для цього потрібно заповнити відповідну форму, вказавши адрессу та дату прибуття, не раніше сьогоднішньої дати.

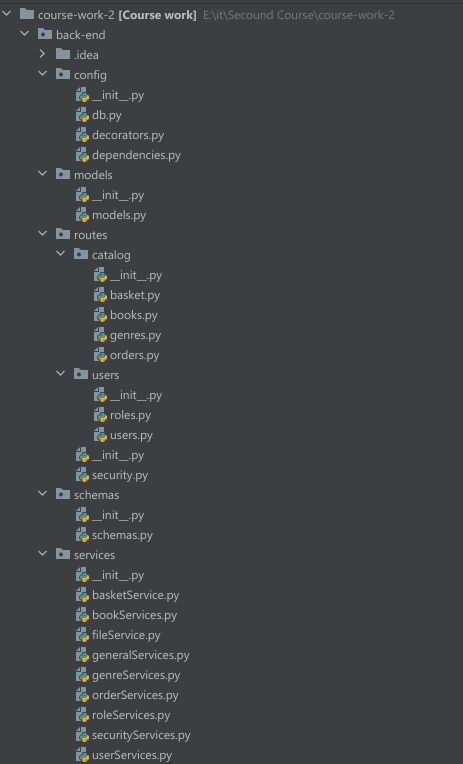
Для корректонсті обробки даних реалізована подвійна валідація, як бекенді так і на фронтенді. При будь-якій віправці форми висвітлбється вікно з оповіщення про статус успішності дії. При редагування дані автоматично заповнюють відповідні поля, для облегчення їх зміни.

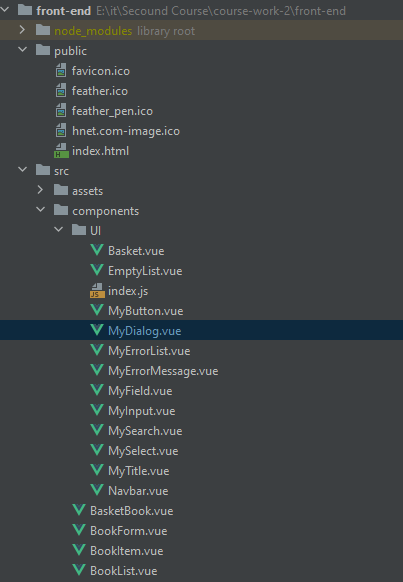
Основним алгоритмом кожної реалізованої функції є:

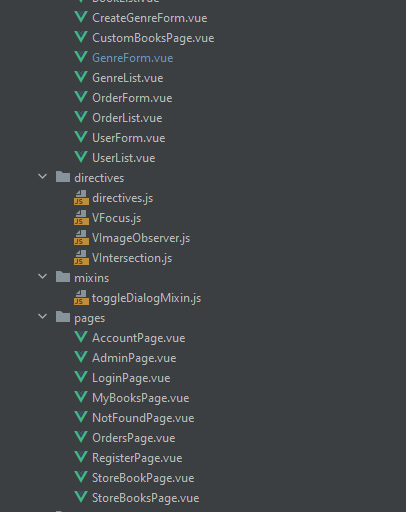
отримання даних на фронтенді -> створення запиту -> передача запиту на бекенд -> обробка отриманих даних у відповідних сервісах(взаємодія з БД) -> передача оброблених даних на фронтенд.

## **2.3 Розробка програмного забезпечення**

Структура рішення проекту :





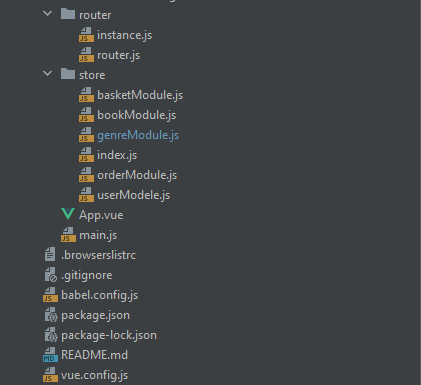


Рисунок 2.2 – Структура проекту

Більша частина програмного забезпечення розробляється відповідно до алгоритму описаного вище, тому розглянемо цей принцип на прикладі реалізованих методів книжок. Методи книжок розмістимо в файлі bookServises, що згодом будуть використовуватись в books-routes, що надають можливість приймати запити з фроненду.

Розглянемо приклад коду створення книжки:

*def* create\_book(*db*: Session, *model*: schemas.BookCreate, *current\_user*: models.User) -> Any:  
 expression = \_model.ISBN == *model*.ISBN  
 generalServices.check\_in\_use\_expression(db=*db*, model=\_model, expression=expression)  
 book = \_model(  
 name=*model*.name,  
 author=*model*.author,  
 content=*model*.content,  
 price=*model*.price,  
 owner=*current\_user*,  
 count=*model*.count,  
 publication\_date=*model*.publication\_date,  
 ISBN=*model*.ISBN  
 )  
 *db*.add(book)  
 *db*.commit()  
 *return* book.id

Маємо метод для створення книжки, який повертає id новостворенної книги та приймає базу даних, з якою ми працюємо, схему з даними та поточного користувача. Використовуємо дані, які передаються в метод, та створюємо книгу. Книга додається в базу, що дає змогу отримати її id згодом.

Розглянемо роут для даногго методу:

@router.post('', response\_model=int)  
*async def* create\_book(*bookCreate*: schemas.BookCreate,*db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 *return* bookServices.create\_book(db=*db*, model=*bookCreate*, current\_user=*current\_user*)

Як модемо бачити код методу не великий, тому що весь функціонал винесений в сервіс.

Далі перейдемо до створення запиту з фронтеду:

async createBook({*state*, *commit*, *rootState*, *rootGetters*}) {  
 await *commit*('setFormLoading', true)  
 *rootState*.errors = []  
 const book = *state*.book  
 book.publication\_date = new *Date*(book.publication\_date.toLocaleString()).toISOString()  
 await instance  
 .post(`${*state*.defaultRoot}`, book, {headers: *rootGetters*.getHeaders})  
 .then(*response* => {  
 book.id = *response*.data

book.image = 'default.png'  
 *commit*('pushBook', book)  
 *commit*('clearBook')  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setFormLoading', false)  
},

Тут ми можемо бачити формування запиту, де instance містить основний шлях до нашого API

export const *instance* = *axios*.create({  
 baseURL: 'http://192.168.0.105:8000/api/',  
});

, що дозволяє уникнути дубляції коду. Виконується метод post, який відпраляє post запит на сервер та приймає як параметри частину шляху, в даному випадку defaultRoot(‘/books’), book – обєкт з даними про книгу config, який містить в headers токен доступу. Далі отриманий результат виконаня обробляється в then(), де ми виконуємо даданя даних до моделі книги та додаємо її в список книг, які показуються користувачу.

Інші запити виконуються по схожому алгоритму, тому вважаю надмірним демонтрувати їх код.

Також важливим компонетом веб додатку є аутентифікація, реалізована на основі JWT. Роглянемо її код:

@router.post("/token", response\_model=schemas.Token)  
*async def* login\_for\_access\_token(*form\_data*: OAuth2PasswordRequestForm = Depends(), *db*: Session = Depends(get\_db)):  
 user = authenticate\_user(db=*db*, email=*form\_data*.username, password=*form\_data*.password)  
 *if not* user:  
 *raise* HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.email, "role": user.role.name}, expires\_delta=access\_token\_expires  
 )  
 *return* {"access\_token": access\_token, "token\_type": "bearer"}

Вище приведений код роута для аутентифікації, який приймає форму з даними. В методі authenticate\_user виконується аутентифікація і якщо вона не успіша користувачу передається повідомлення про помилку.

Розглянемо дедальніше цей метод:

*def* authenticate\_user(*db*: Session, *email*: str, *password*: str) -> Any:  
 model = models.User  
 expression = model.email == *email* user = generalServices.get\_by\_expression(db=*db*, model=model, expression=expression)  
 *if not* verify\_password(*password*, user.password):  
 *return False  
 return* user

В даному методі ми звертаємося до бази даних і перевіряємо на співпадіння паролі та повертаємо відповідний реультат.

Після того як ми отримали результат аутентифікації, при успішності ми створюємо токен доступута записуємо в нього дані про email, role та дату закінчення токену. І в фіналі повертаємо на фронтенд результат виконання методу, а саме новостворений токен доступу.

## **Висновки до 2 розділу :**

Спроектовано загальну схему програми. Визначено та детально описано основні можливості програми. Пояснено алгоритми найважливіших методів інтернет-магазину та приведено код їх основних частин. Приведено приклади коду деяких методів. Роз’яснено деякі нюанси роботи програми. Показано структуру проекта.

# РОЗДІЛ 3 ОПИС РОБОТИ З ДОДАТКОМ

## **3.1 Опис роботи з додатком (Опис інтерфейсу)**

Після запуску програми з’являється головна сторінка (Рис 3.1), де користувач бачить список книг та може тільки їх проглядати.

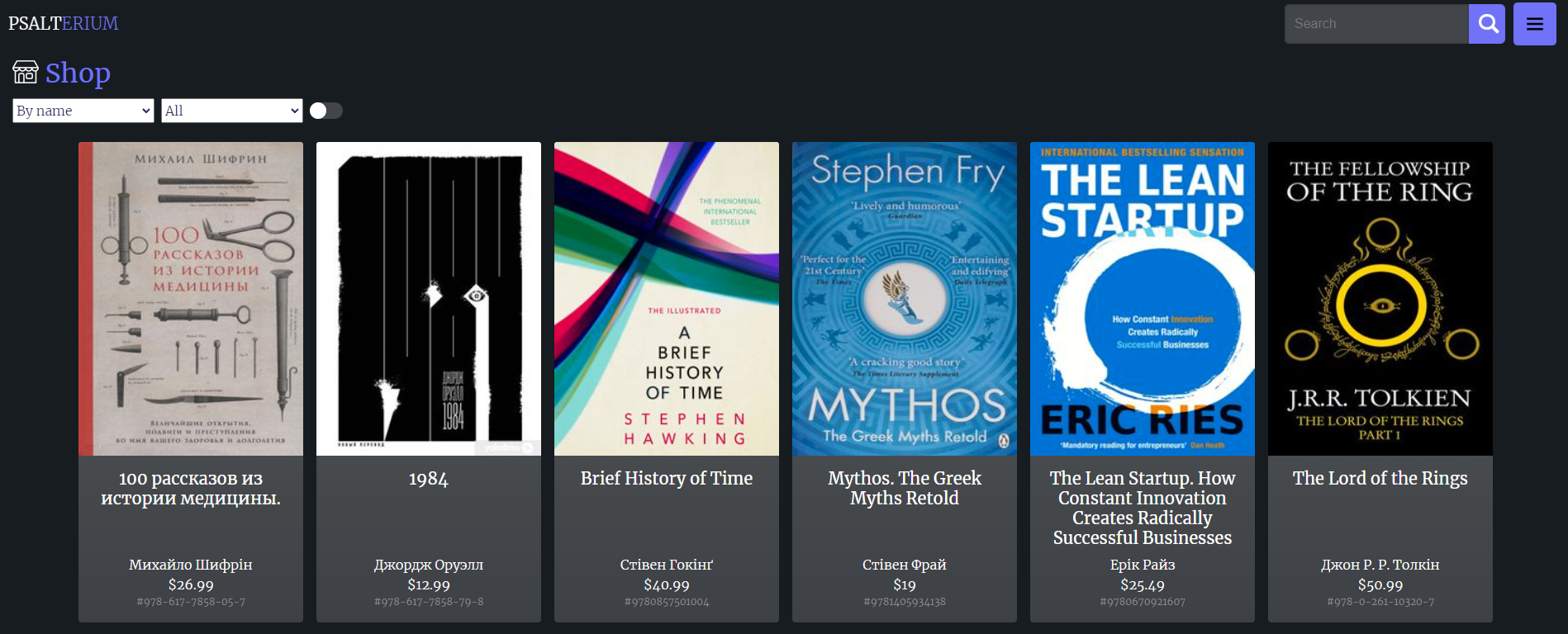


Рис 3.1 Головна сторінка

Головна сторінка має функції переходу до конкретних книжок, сортування та пошуку. Також реалізована динамічна пагінація у виді бескінечної ленти. У верхньому правому куті розміщений пошук та меню, де користувач може побачити наявні сторінки.

Перейдемо до конкретного алгоритму роботи х веб додатком.

Спочатку користувач обирає книгу, наприклад, книгу “Чистий код”. При натиску на карточку з відповідною книгою, він переходить на сторінку книги (Рис 3.3), де він може прочитати інформацію про книгу більш детально та додати деяку кількість книг до власної корзини. Після успішного додання книги до корзини, користувач бачить повідомлення:

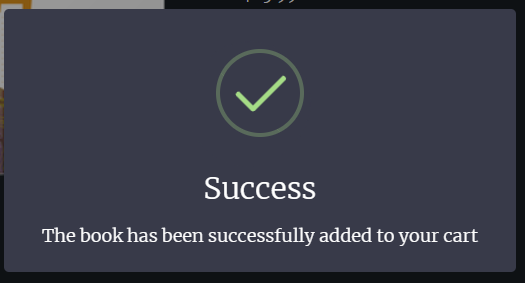


Рис 3.2 Сторінка конкретної книги

Далі користувач може переглядати інші книги та додавати до корзини або оформити замовлення.

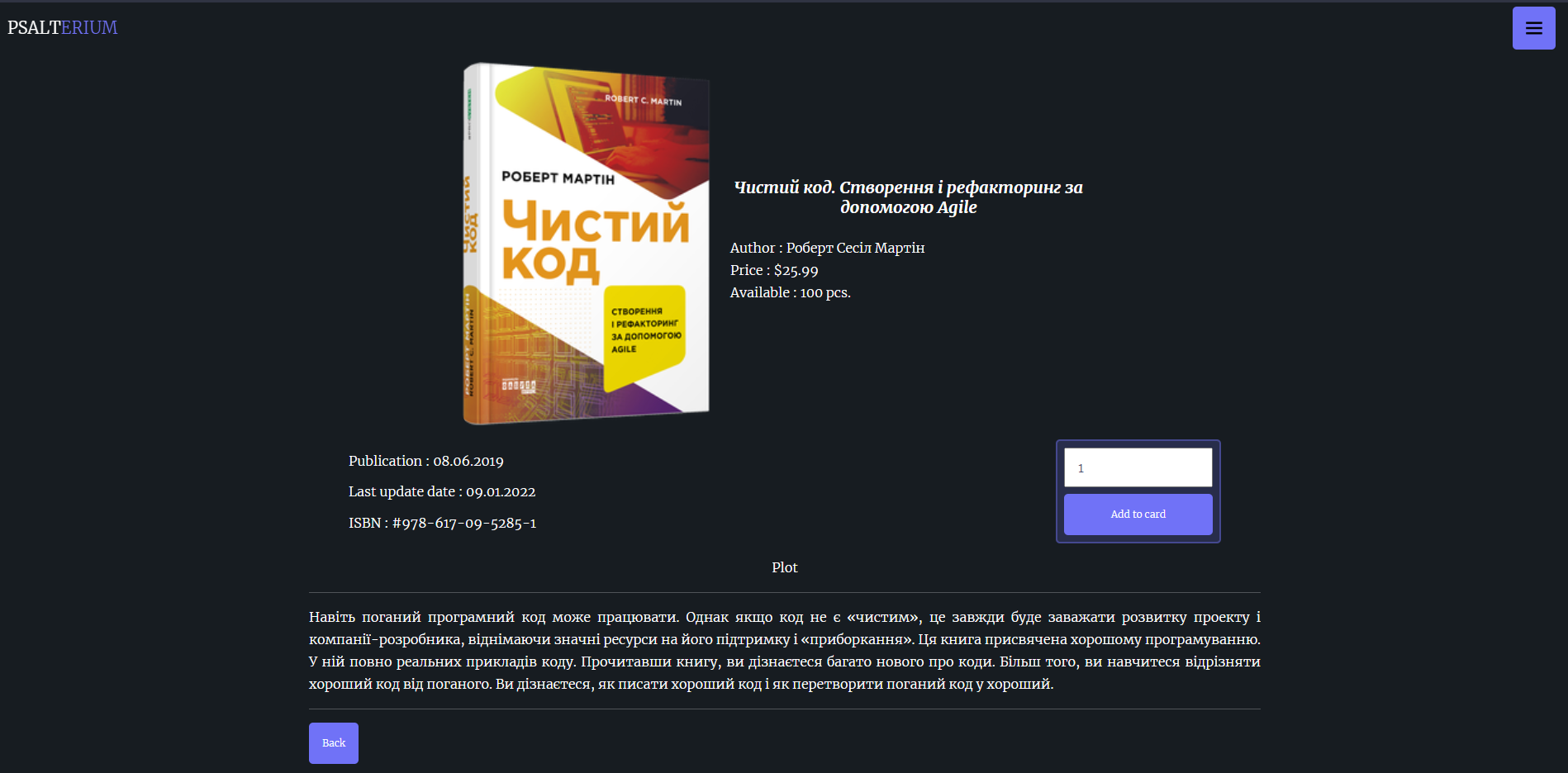


Рис 3.3 Сторінка конкретної книги

Для прикладу оберемо ще одну книгу(“1984”). Додали до корзини 3 екземпляри. Далі користувач переходить до корзини для оформлення замовлення(Рис 3.4). Як можемо бачити, тут відображаються всі нами обрані книги.

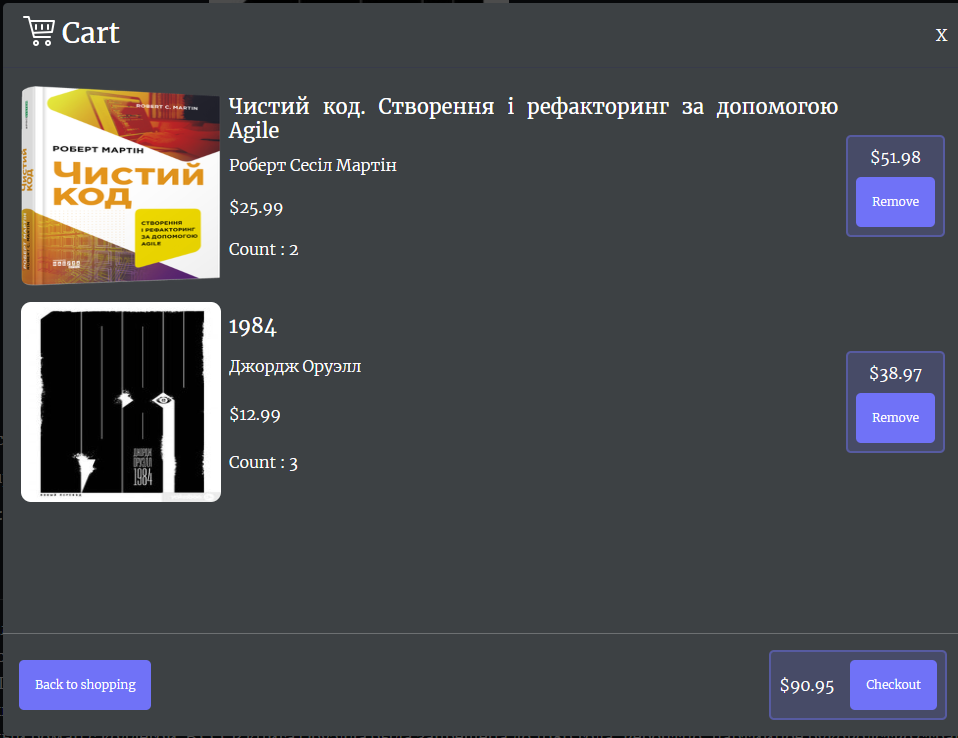


Рис 3.4 Корзина

Також користувач може видалити з корзини книгу, натиснувши на кнопку зліва від книги. Над кнопкою також відображена ціна за всі екземпляри книги. Якщо користувач впевнений, що йому саме ці книги потрібні, він переходить до оформлення замовлення(Рис 3.5). Після оформлення форми замовлення додається в список замовлень(Рис 3.6).

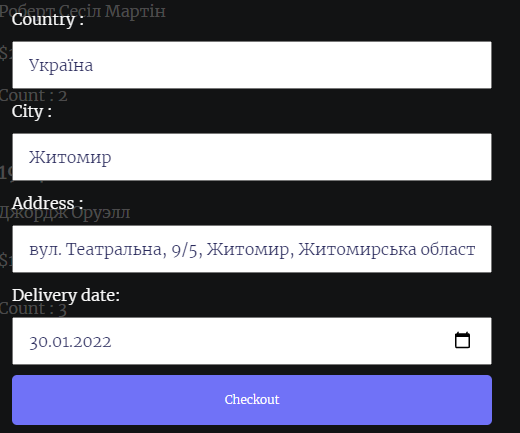


Рис 3.5 Форма оформлення замовлення

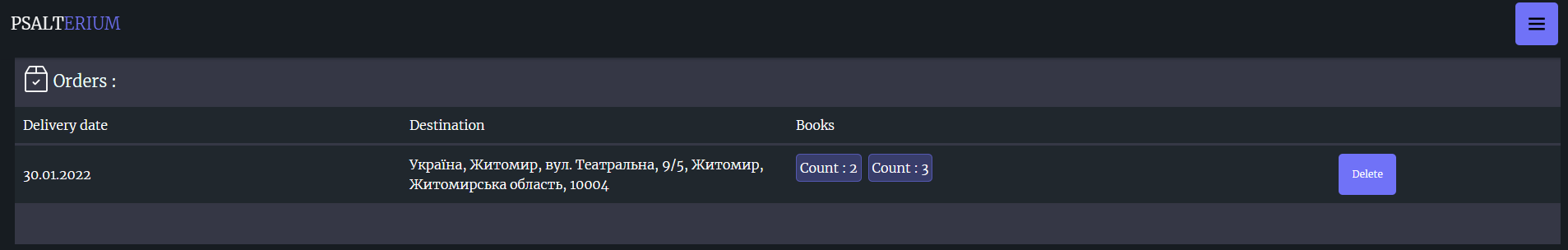


Рис 3.6 Список замовлень

На сторінці список замовлень користувач може відміняти замовлення та переглянути замовлені книги при кліку на підсвічені елементи, які ведуть на сторінки відповідних книжок. На даній сторінці адміну будуть показуватися всі замовлення, а звичайному користувачу тільки його власні.

Далі перейдемо до дадаткової можливості інтернет-магазину – виставлення на продаж власних книг.

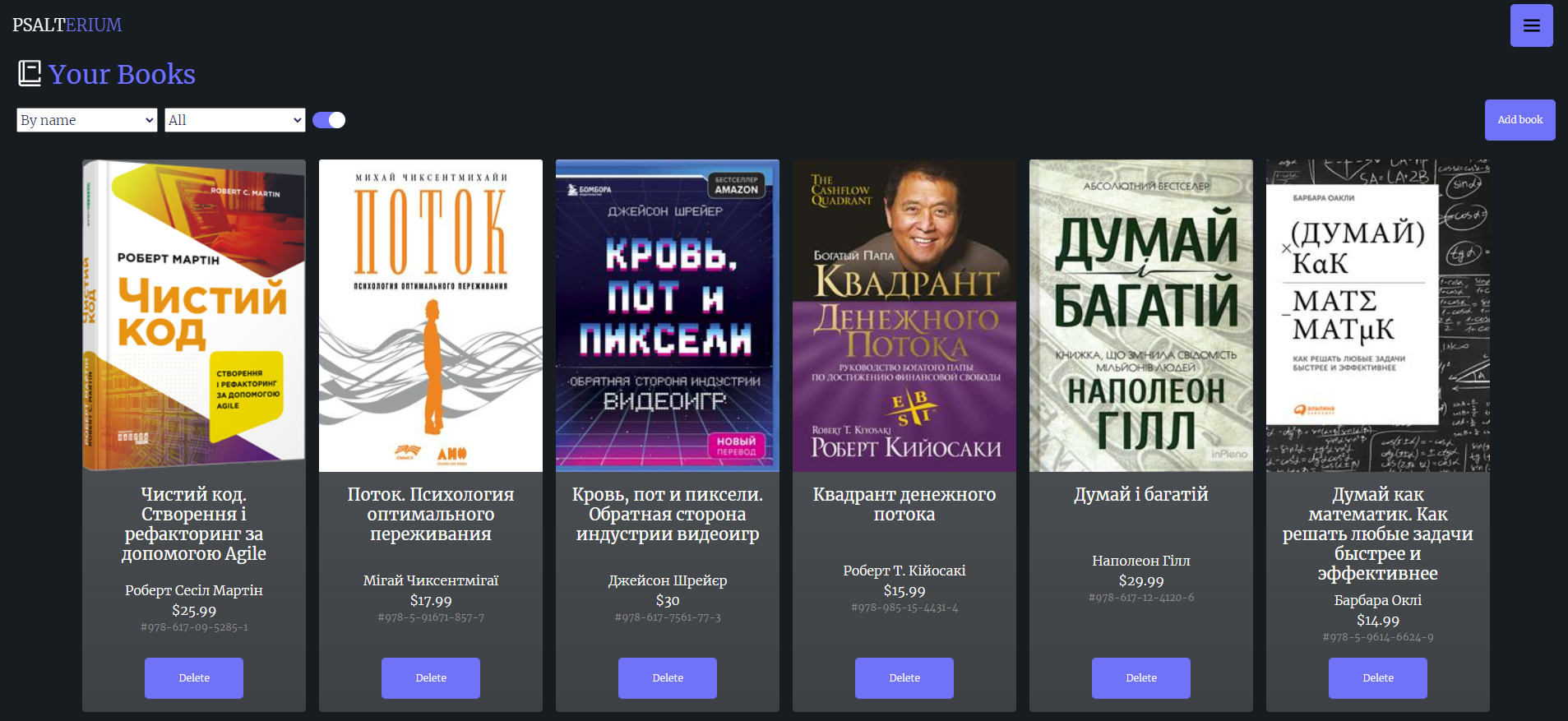


Рис 3.7 Сторінка книг виставлених на продаж користувачем

На даній сторінці у верхньому правому куті розміщена кнопка, яка відкриває форму для книги(Рис. 3.8). Коли користувач заповнить цю форму, книга відобразиться на сторінці та додасться до магазину, де інший користувач зможе її замовити.

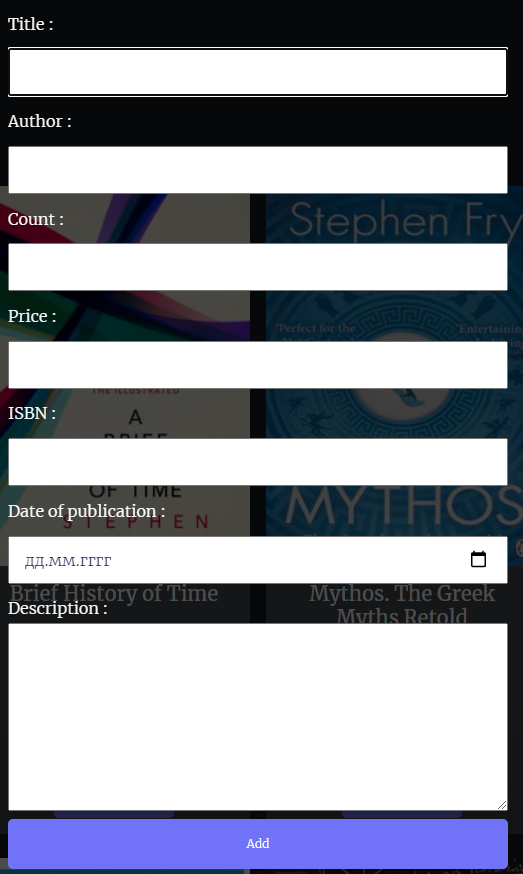


Рис 3.7 Форма для виставлення книг на продаж

Після виставлення книги, її користувач може перейти на неї і побачити додатковий функціонал, а саме можливість зміни ланих про книгу та задати їй жанри.

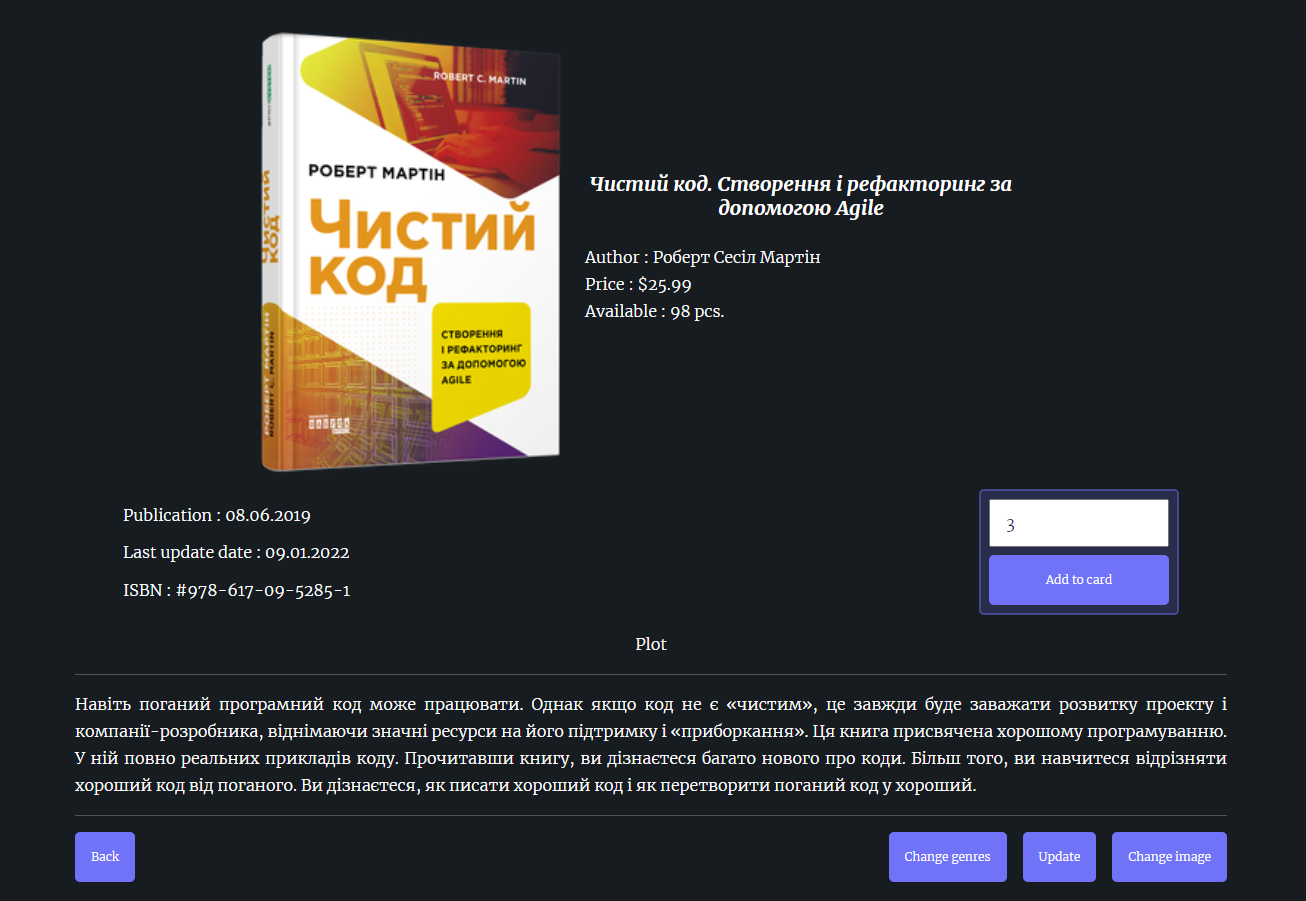


Рис 3.8 Сторінка книги з розширеним функціоналом

Користувач може перейти на власну сторінку(Рисю 3.9) і змінити про себе дані та пароль.

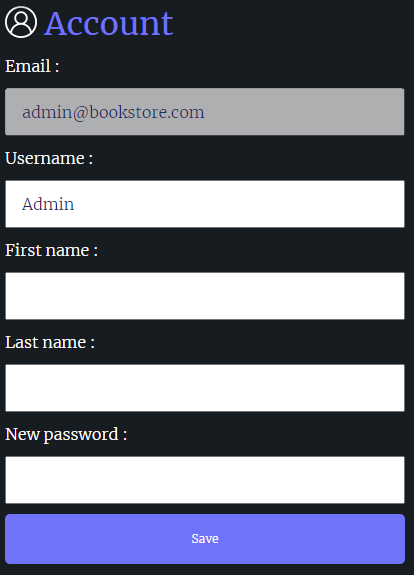


Рис 3.9 Форма для зміни інформації про користувача

Також не можна не розглянути функціонал який доступний тільки для адмінів, а саме сторінку адміністрації(Рис. 3.10).

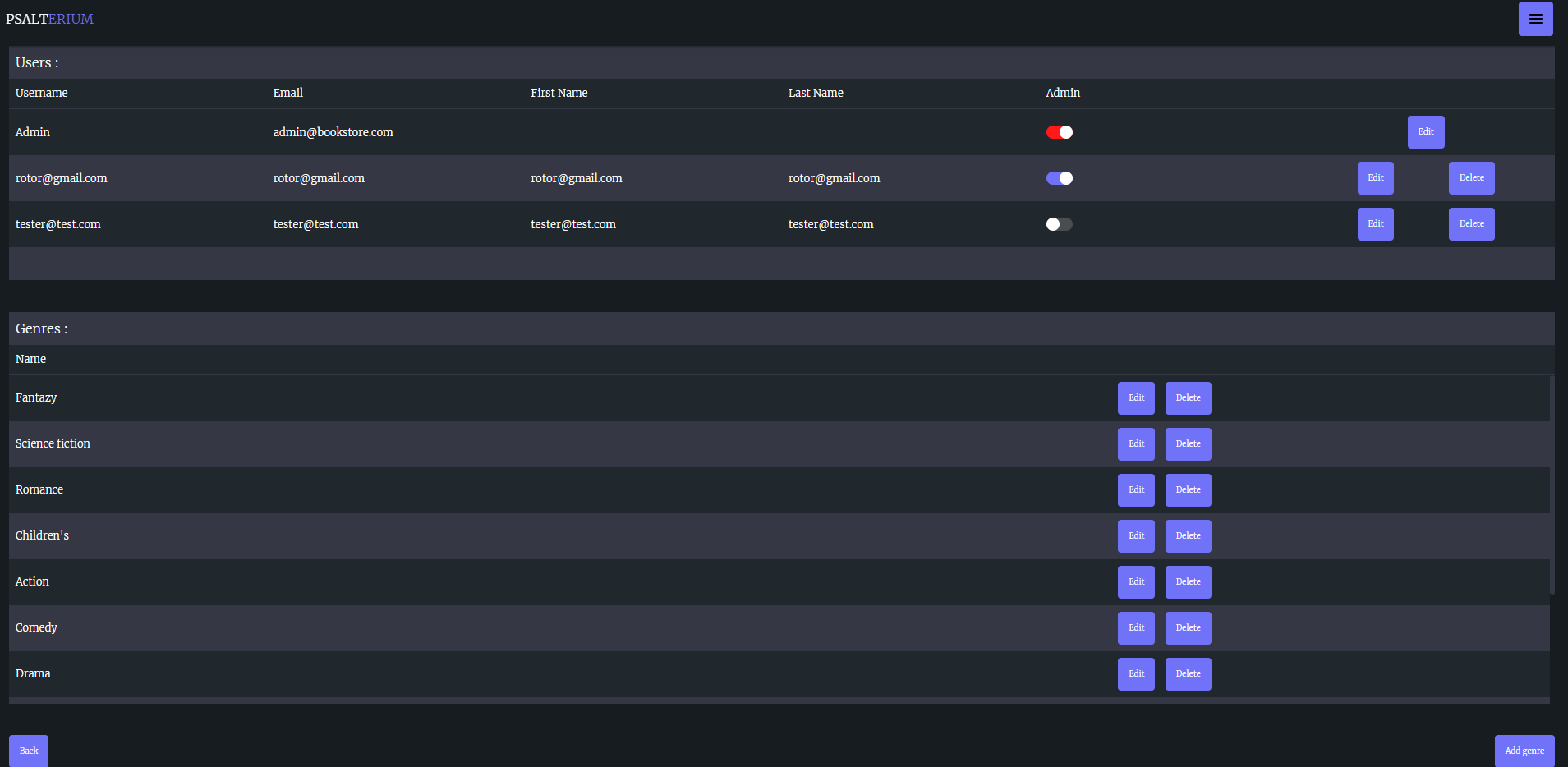


Рис 3.10 Сторінка адміністрації

На сторінку адміністрації мають доступ тільки адміністратори, як не дивно. На даній сторінці адміністратор може переглядати та змінювати дані про користувачів та жанри. Також має можливість видаляти і користувачів і жанри. Користувачі створюються при реєстрації, а для створення жанрів зліва знизу розміщена кнопка для відкриття форми створення жанру(Рис 3.11). Також адмін має можливість надати права адміна іншому користувачу.

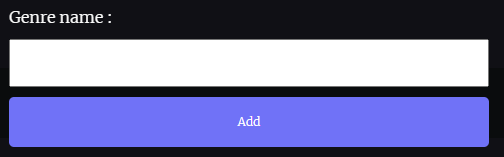


Рис 3.10 Форма створення жанру

Після кожної успішної дії користувач отримує повідомлення про успішність виконання. Розглянемо на прикладі створення нового жанру.

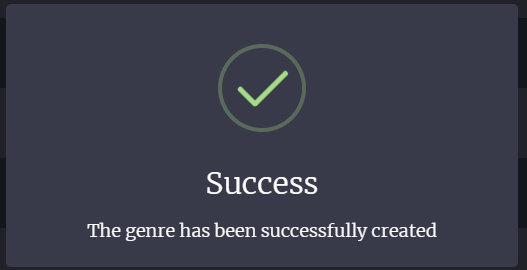


Рис 3.11 Повідомлення про успішне створення жанру

## **Тестування веб-додатку**

Веб-додаток відображає попередження про помилку при при введенні даних для зручного користування користувачем. Наприклад, при спробі створення “пустого” користувача відображаються попередження про обов’язковість полів(Рис 3.12).

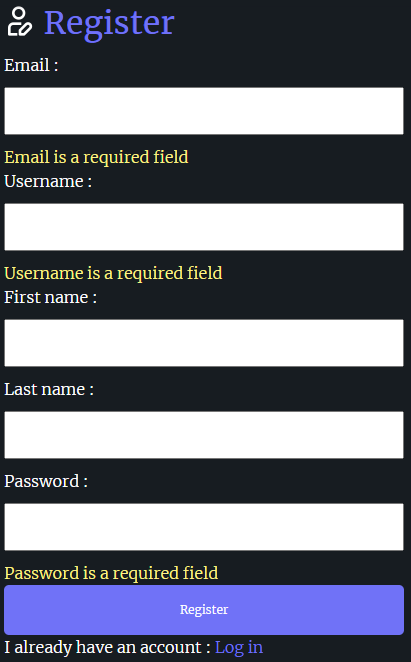


Рис 3.12 Повідомлення про необхідність заповнення

При вводі некоректного значення, з’являється попередження(Рис 3.13)

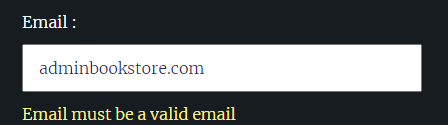


Рис 3.13 Повідомлення про некоректність вводу

При вводі даних які вже існують і не можуть дублюватися, з’являється попередження про існування(Рис 3.14)

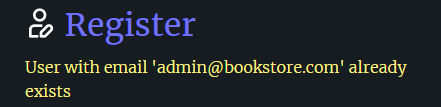


Рис 3.14 Повідомлення про існування користувача

При відправці форми з некоректними даними аутентифікації, з’являється попередження некоректність(Рис 3.15)

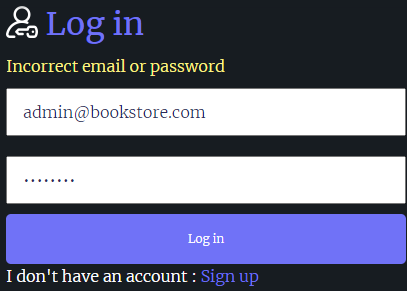


Рис 3.15 Повідомлення некоректність email або пароля

Подібна валідація реалізована на всіх формах

## **Висновки до 3 розділу :**

Було детально описано роботу програмного додатку, користувацький інтефейс, очікувані дії користувача та їх результати. Також було розписано моменти уникнення помилок завдяки допоміжним перевіркам при введенні даних або при роботі з зображеннями. Завдяки цим перевіркам вдалося уникнути частих та серйозних помилок. Наведено приклади обробки помилок.

# **ВИСНОВКИ**

Під час написання даного курсового проекту було отримано навички роботи з розробки веб-додатку.

В першому розділі курсового проекту, було проаналізовано знайдені інтернет-магазини зі схожим функціоналом та інтерфейсом, та було сформовано уявлення про інтернет-магазин, що розроблюється. Було виявлено основні потреби для написання даного веб-додатку.

В другому розділі курсового проекту, було проведено розробку функціональних і загальних частин веб-додатку. При розробці функціонального алгоритму, розробили уявлення про методи розробки програмного коду. На заключному етапі, розроблено його. В результаті, отримано готовий програмний продукт, який виконуває всі заплановані функції і задачі.

В третьому розділі курсового проекту було описана методи користування створеним інтернет-магазином, були пояснені і продемонстровані всі створені елементи інтерфейсу. Також було проведено тестування, тобто інтернет-магазин було запущено в реальних умовах її користування. В ході тестування проблем не виникало.

В результаті виконання курсового проекту отримано інтернет-магазин книжок, який повністю відповідає запланованому функціоналу, умовам та вимогам, поставленим на початку проектування та привищує його. Інтернет-магазин є зрозумілим та зручним в користуванні, адаптивним дял різних пристроїв та швидкодіючим.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Фрэйн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрэйн ; [перевод с английского В. Черник]. - Санкт-Петербург [и др.] : Питер, 2014. - 298 с.
2. Лоусон Б., Шарп Р. Изучаем HTML5. Библиотека специалиста. 2-е изд. – СПб.: Питер, 2012. – 304 с
3. Флэнаган Д. JavaScript. Подробное руководство, 6-е издание – Пер. с англ. – СПб: Символ-Плюс, 2012. – 1080 с.
4. А.Зандстра, Ф.Матт. PHP Objects, Patterns, and Practice, 2013. Режим доступу: https://www.apress.com/la/book/9781430260325
5. Дмитрий Кирсанов. “Веб-дизайн” [електронний ресурс] - 2006. Режим доступу: <http://web-diz.com.ua/skachat/veb-dizayn--dmitriy-kirsanov-skachat-besplatno/>
6. Асинхронное программирование . [Електронний ресурс] – Режим доступу: <https://metanit.com/sharp/tutorial/13.3.php>
7. Тим Кедлек. “Адаптивный дизайн. Делаем сайты для любых устройств” [електронний ресурс] - 2013. Режим доступу: <https://www.ozon.ru/context/detail/id/21098804/>
8. Плюси і мінуси обєктно-орієнтованого програмування [електронний ресурс] - 2015. Режим доступу: <http://damp.biz/plyusi-i-minusi-obyektno-oriyentovanogoprogramuvannya/>
9. SQL [електронний ресурс] - 2016. Режим доступу: <https://uk.wikibooks.org/wiki/SQL>
10. Создание SQL запросов [електронний ресурс] - 2016. Режим доступу: <https://www.politerm.com/zuludoc/spatial_query_sql.htm#_sql_overview>
11. Документація Fast API[електронний ресурс]. Режим доступу: <https://fastapi.tiangolo.com/>
12. Документація Vue js [електорнний ресурс]. Режим доступу: https://v3.vuejs.org/

# **ДОДАТКИ**

Додаток А

**Програмний код:**

**Back-end :**

***Setup.py :***

*import* uvicorn  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 uvicorn.run(app='main:app', host='0.0.0.0', port=8000, reload=*True*)

***Main.py :***

*from* fastapi *import* FastAPI  
*from* routes *import* router  
*from* config *import* dependencies  
*from* fastapi.middleware.cors *import* CORSMiddleware  
*from* starlette.types *import* ASGIApp, Scope, Receive, Send  
  
  
  
app = FastAPI()  
  
origins = [  
 "\*"  
]  
  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=origins,  
 allow\_credentials=*True*,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
*class* SimpleASGIMiddleware:  
 *def \_\_init\_\_*(self, *app*: ASGIApp):  
 self.app = *app  
  
 async def \_\_call\_\_*(self, *scope*: Scope, *receive*: Receive, *send*: Send):  
 *await* self.app(*scope*, *receive*, *send*)  
 client = *scope*["client"]  
 print(f"[CLIENT]: {client}")  
  
  
app.add\_middleware(SimpleASGIMiddleware)  
  
dependencies.create\_database()  
app.include\_router(router)

***Exceptions :***

*from* typing *import* Any  
*from* fastapi *import* HTTPException  
  
*def* CustomNotFoundException(*entity*: Any, *key*: str, *value*: Any):  
 *raise* HTTPException(status\_code=404, detail=f"{*entity*.\_\_name\_\_} with {*key*} '{*value*}' not found")  
  
*def* CustomExistException(*entity*: Any, *key*: str, *value*: Any):  
 *raise* HTTPException(status\_code=400, detail=f"{*entity*.\_\_name\_\_} with {*key*} '{*value*}' already exists")  
  
*def* CustomAccessForbiddenException():  
 *raise* HTTPException(status\_code=403, detail=f"Access is forbidden! You are not an admin")  
  
*def* CustomCountException(*entity*: Any, *key*: str, *value*: Any):  
 *raise* HTTPException(status\_code=400, detail=f"Not enough {(*entity*.\_\_name\_\_).lower()}s with {*key*} {*value*} in stock")

***Back-end services :***

*userServices.py :*

*from* varname *import* nameof  
  
*from* Exceptions *import* CustomExistException  
*from* models *import* models  
*from* schemas *import* schemas  
*from* config.db *import* Session  
*from* . *import* generalServices, securityServices, bookServices  
  
\_model = models.User  
\_role\_model = models.Role  
  
*def* check\_email\_in\_use(*db*: Session, *email*: str) -> \_model:  
 *return db*.query(\_model).filter(\_model.email == *email*).first()  
  
*def* create\_user(*db*: Session, *model*: schemas.UserCreate) -> int:  
 \_ = check\_email\_in\_use(db=*db*, email=*model*.email)  
 *if* \_:  
 CustomExistException(entity=\_model, key=nameof(*model*.email), value=*model*.email)  
  
 user\_role = generalServices.get\_by\_expression(db=*db*, model=\_role\_model, expression=\_role\_model.name == 'User')  
  
 user = \_model(  
 email=*model*.email,  
 username=*model*.username,  
 first\_name=*model*.first\_name,  
 last\_name=*model*.last\_name,  
 password=securityServices.get\_password\_hash(*model*.password),  
 role = user\_role  
 )  
  
 *db*.add(user)  
 *db*.commit()  
 *return* user.id  
  
  
*def* update\_user(*db*: Session, *model*: schemas.UserCreate, *expression*: bool):  
 user = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
 user.username = *model*.username  
 user.first\_name = *model*.first\_name  
 user.last\_name = *model*.last\_name  
 *if model*.password:

user.password = securityServices.get\_password\_hash(*model*.password)  
 *db*.commit()  
  
*def* change\_user\_role(*db*: Session, *user\_id*: int, *role\_name*: str):  
 user = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=\_model.id == *user\_id*)  
 role = generalServices.get\_by\_expression(  
 db=*db*,  
 model=\_role\_model,  
 expression=\_role\_model.name == *role\_name*.lower().capitalize()  
 )  
 user.role = role  
 *db*.commit()  
  
*def* delete\_user(*db*: Session, *id*: int):  
 user = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=\_model.id == *id*)  
 *for* book *in* user.books:  
 bookServices.delete\_book(db=*db*, id=book.id)  
 generalServices.delete(db=*db*, model=\_model, id=user.id)

*securityServices.py :*

*from* datetime *import* timedelta, datetime  
*from* typing *import* Optional, Any  
*from* jose *import* jwt  
*from* passlib.context *import* CryptContext  
  
*from* config *import* SECRET\_KEY, ALGORITHM  
*from* config.db *import* Session  
*from* models *import* models  
*from* . *import* generalServices  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
*def* verify\_password(*plain\_password*, *hashed\_password*) -> bool:  
 *return* pwd\_context.verify(*plain\_password*, *hashed\_password*)  
  
*def* get\_password\_hash(*password*) -> str:  
 *return* pwd\_context.hash(*password*)  
  
*def* authenticate\_user(*db*: Session, *email*: str, *password*: str) -> Any:  
 model = models.User  
 expression = model.email == *email* user = generalServices.get\_by\_expression(db=*db*, model=model, expression=expression)  
 *if not* verify\_password(*password*, user.password):  
 *return False  
 return* user  
  
*def* create\_access\_token(*data*: dict, *expires\_delta*: Optional[timedelta] = *None*) -> Any:  
 to\_encode = *data*.copy()  
 *if expires\_delta*:  
 expire = datetime.utcnow() + *expires\_delta  
 else*:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 *return* encoded\_jwt

*roleServices.py :*

*from* varname *import* nameof  
  
*from* Exceptions *import* CustomExistException  
*from* models *import* models  
*from* schemas *import* schemas  
*from* config.db *import* Session  
*from* services *import* generalServices  
  
\_model = models.Role  
  
*def* get\_role(*db*: Session, *name*: str) -> \_model:  
 *return db*.query(\_model).filter\_by(name = *name*).first()  
  
  
*def* create\_role(*db*: Session, *model*: schemas.RoleCreate) -> int:  
 \_ = get\_role(db=*db*, name=*model*.name)  
 *if* \_:  
 CustomExistException(entity=\_model, key=nameof(*model*.name), value=*model*.name)  
  
 role = \_model(name=*model*.name)  
 *db*.add(role)  
 *db*.commit()  
 *return* role.id  
  
  
*def* update\_role(*db*: Session, *model*: schemas.RoleCreate, *expression*: bool):  
 \_ = get\_role(db=*db*, name=*model*.name)  
 *if* \_:  
 CustomExistException(entity=\_model, key=nameof(*model*.name), value=*model*.name)  
  
 role = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
 role.name = *model*.name  
 *db*.commit()

*orderServices.py :*

*from* datetime *import* datetime  
*from* sqlalchemy *import* and\_  
*from* config.db *import* Session  
*from* schemas *import* schemas  
*from* services *import* generalServices  
*from* models *import* models  
\_model = models.Order  
  
  
*def* create\_order(*db*: Session, *model*: schemas.OrderCreate, *current\_user*: models.User) -> int:  
 order = \_model(  
 deliver\_date=*model*.deliver\_date,  
 country=*model*.country,  
 city=*model*.city,  
 address=*model*.address,  
 user\_id=*current\_user*.id,  
 )  
 *db*.add(order)  
 expression = and\_(models.OrderBook.order\_id == 0, models.OrderBook.consumer == *current\_user*)  
 order\_books = generalServices.get\_all\_without\_limit(db=*db*, model=models.OrderBook, expression=expression)  
 *for* order\_book *in* order\_books:  
 order\_book.order = order  
 *db*.commit()  
 *return* order.id  
  
  
*def* delete\_order(*db*: Session, *id*: int):  
 order = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=\_model.id == *id*)  
 expression = models.OrderBook.order == order  
 order\_books = generalServices.get\_all\_without\_limit(db=*db*, model=models.OrderBook, expression=expression)  
 *for* order\_book *in* order\_books:  
 book = generalServices.get\_by\_expression(db=*db*, model=models.Book, expression=models.Book.id == order\_book.book\_id)  
 book.count += order\_book.count  
 *db*.delete(order\_book)  
 *db*.commit()  
 generalServices.delete(db=*db*, model=\_model, id=*id*)  
 *db*.commit()

*genreServices.py :*

*from* typing *import* Any  
  
*from* varname *import* nameof  
  
*from* Exceptions *import* CustomExistException  
*from* models *import* models  
*from* schemas *import* schemas  
*from* config.db *import* Session  
*from* services *import* generalServices  
*from* services.generalServices *import* check\_in\_use\_expression  
  
\_model = models.Genre  
  
*def* create\_genre(*db*: Session, *model*: schemas.GenreCreate) -> int:  
 expression = \_model.name == *model*.name  
 check\_in\_use\_expression(db=*db*, model=\_model, expression=expression)  
 genre = \_model(name = *model*.name)  
 *db*.add(genre)  
 *db*.commit()  
 *return* genre.id  
  
  
*def* update\_genre(*db*: Session, *model*: schemas.GenreCreate, *expression*: bool):  
 genre = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
 *if not model*.name == genre.name:  
 check\_in\_use\_expression(db=*db*, model=\_model, expression=\_model.name == *model*.name)  
 genre.name = *model*.name  
 *db*.commit()

*generalServices.py :*

*from* typing *import* List, Any  
  
*from* models *import* models

*from* Exceptions *import* CustomNotFoundException, CustomExistException, CustomCountException  
*from* varname *import* nameof  
  
*from* config.db *import* Session  
  
*def* get\_all(*db*: Session, *model*: Any, *skip*: int, *limit*: int) -> List[Any]:  
 *return db*.query(*model*).order\_by(*model*.id).offset(*skip*).limit(*limit*).all()  
  
*def* get\_all\_with\_expression(*db*: Session, *model*: Any, *skip*: int, *limit*: int, *expression*: bool) -> List[Any]:  
 *return db*.query(*model*).filter(*expression*).order\_by(*model*.id).offset(*skip*).limit(*limit*).all()  
  
*def* get\_all\_without\_limit(*db*: Session, *model*: Any, *expression*: Any) -> List[Any]:  
 *return db*.query(*model*).order\_by(*model*.id).filter(*expression*).all()  
  
*def* get\_by\_expression(*db*: Session, *model*: Any, *expression*: Any) -> Any:  
 entity = *db*.query(*model*).filter(*expression*).first()  
 *if not* entity:  
 key = *expression*.left.key  
 value = *expression*.right.value  
 CustomNotFoundException(entity=*model*, key=key, value=value)  
 *return* entity  
  
*def* delete(*db*: Session, *model*: Any, *id*: int):  
 entity = get\_by\_expression(db=*db*, model=*model*, expression=*model*.id == *id*)  
 *db*.delete(entity)  
 *db*.commit()  
  
*def* check\_in\_use\_expression(*db*: Session, *model*: Any, *expression*: Any):  
 \_ = *db*.query(*model*).filter(*expression*).first()  
 *if* \_:  
 key = *expression*.left.key  
 value = *expression*.right.value  
 CustomExistException(entity=*model*, key=key, value=value)  
  
*def* check\_in\_warehouse(*db*: Session, *model*: Any, *id*: int, *count*: int):  
 expression = *model*.id == *id* entity = get\_by\_expression(db=*db*, model=*model*, expression=expression)  
 *if*(entity.count < *count*):  
 CustomCountException(entity=*model*, key=nameof(*id*), value=*id*)

*fileServices.py :*

*import* os  
*from* uuid *import* uuid4  
*from* fastapi *import* UploadFile  
*from* fastapi.responses *import* FileResponse  
*from* PIL *import* Image, ImageFile  
  
  
static\_files\_path = os.path.join(os.getcwd(), 'static\\files')  
static\_assets\_path = os.path.join(os.getcwd(), 'static\\assets')  
  
  
*def* save\_file(*file*: UploadFile, *height*: int) -> str:  
 print(*file*.\_\_dict\_\_)  
 \_, ext = *file*.filename.rsplit('.', 1)  
 filename = f'{uuid4()}.{ext}'  
 location = os.path.join(static\_files\_path, filename)  
 img = Image.open(*file*.file)  
 ratio = (*height* / float(img.size[1]))  
 width = int((float(img.size[0]) \* float(ratio)))  
 img = img.resize((width, *height*), Image.ANTIALIAS)  
 type = *file*.content\_type.split('/')[1]  
 img.save(location, type, quality=80, optimize=*True*, progressive=*True*)  
 *return* filename  
  
*def* get\_file(*path*: str) -> FileResponse:  
 full\_path = get\_full\_path(static=static\_files\_path, path=*path*)  
 *if not* os.path.exists(full\_path):  
 full\_path = get\_full\_path(static=static\_assets\_path, path='default.png')  
 *return* FileResponse(full\_path)  
  
*def* delete\_file(*path*: str):  
 full\_path = get\_full\_path(static=static\_files\_path,path=*path*)  
 *if* os.path.exists(full\_path):  
 os.remove(full\_path)  
  
  
*def* get\_full\_path(*static*: str, *path*: str):  
 *return* os.path.join(*static*, *path*)

*bookServices.py :*

*import* datetime  
*from* typing *import* List, Any  
*from* sqlalchemy *import* and\_  
*from* fastapi *import* UploadFile  
*from* fastapi.responses *import* FileResponse  
*from* sqlalchemy.sql *import* text  
  
*from* models *import* models  
*from* schemas *import* schemas  
*from* config.db *import* Session  
*from* datetime *import* datetime  
*from* . *import* generalServices, fileService  
  
  
\_model = models.Book  
  
*def* get\_sorted\_books(*db*: Session, *skip*: int, *limit*: int, *query*: str, *reverse*: bool, *genre*: str, *filter\_expression*: bool = *True*):  
 expression = \_model.name  
  
 *if query* == 'author':  
 expression = \_model.author  
 *elif query* == 'price':  
 expression = \_model.price  
 *elif query* == 'count':  
 expression = \_model.count  
 *elif query* == 'date':  
 expression = \_model.publication\_date  
 *elif query* == 'name':  
 expression = \_model.name  
  
 *if reverse*:  
 expression = expression.desc()  
  
 *if genre* == 'all':  
 genre\_expression = *True  
 else*:  
 genre\_entity = generalServices.get\_by\_expression(db=*db*, model=models.Genre, expression=models.Genre.name == *genre*)  
 print(genre\_entity)  
 genre\_expression = \_model.genres.contains(genre\_entity)  
  
 filter = and\_(genre\_expression, *filter\_expression*)  
  
 *return db*.query(\_model).filter(filter).order\_by(expression).offset(*skip*).limit(*limit*).all()  
  
  
*def* create\_book(*db*: Session, *model*: schemas.BookCreate, *current\_user*: models.User) -> int:  
 expression = \_model.ISBN == *model*.ISBN  
 generalServices.check\_in\_use\_expression(db=*db*, model=\_model, expression=expression)  
 book = \_model(  
 name=*model*.name,  
 author=*model*.author,  
 content=*model*.content,  
 price=*model*.price,  
 owner=*current\_user*,  
 count=*model*.count,  
 publication\_date=*model*.publication\_date,  
 ISBN=*model*.ISBN  
 )  
 *db*.add(book)  
 *db*.commit()  
 *return* book.id  
  
  
*def* update\_book(*db*: Session, *model*: schemas.BookCreate, *expression*: Any):  
 book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
  
 book.name = *model*.name  
 book.author = *model*.author  
 book.content = *model*.content  
 book.price = *model*.price  
 book.count = *model*.count  
 book.update\_date = datetime.utcnow()  
 print(book.\_\_dict\_\_)  
 *if not* book.ISBN == *model*.ISBN:  
 expression = \_model.ISBN == *model*.ISBN  
 generalServices.check\_in\_use\_expression(db=*db*, model=\_model, expression=*expression*)  
 book.ISBN = *model*.ISBN  
 *db*.commit()  
  
  
*def* set\_genres(*db*: Session, *genres*: List[schemas.GenreBase], *expression*: Any):  
 book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
 book.genres = [  
 generalServices.get\_by\_expression(  
 db=*db*,  
 model=(model:=models.Genre),  
 expression=model.name == genre.name  
 )  
 *for* genre *in genres*

]  
 *db*.commit()  
  
  
*def* get\_image(*db*: Session, *id*: int) -> FileResponse:  
 book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=\_model.id == *id*)  
 print(book.image)  
 *return* fileService.get\_file(book.image)  
  
  
*def* get\_image\_by\_name(*name*: str) -> FileResponse:  
 *return* fileService.get\_file(*name*)  
  
  
*def* change\_image(*db*: Session, *image*: UploadFile, *expression*: Any) -> str:  
 book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=*expression*)  
 print(book.image)  
 fileService.delete\_file(book.image)  
 book.image = fileService.save\_file(*image*, 300)  
 *db*.commit()  
 print(book.image)  
 *return* book.image  
  
  
  
*def* delete\_book(*db*: Session, *id*: int):  
 book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression= \_model.id == *id*)  
 expression = models.OrderBook.book\_id == book.id  
 orders = generalServices.get\_all\_without\_limit(db=*db*, model=models.OrderBook, expression=expression)  
 *for* order *in* orders:  
 *if*(order.order\_id == 0):  
 *db*.delete(order)  
 *else*:  
 expression = models.Order.id == order.order\_id  
 \_order = generalServices.get\_by\_expression(db=*db*, model=models.Order, expression=expression)  
 print(\_order.books)  
 \_order.books.remove(order)  
 print(\_order.books)  
 *db*.delete(order)  
 *if not* \_order.books:  
 *db*.delete(\_order)  
 *db*.delete(book)  
 *db*.commit()

*basketServices.py :*

*from* datetime *import* datetime  
*from* sqlalchemy *import* and\_  
*from* config.db *import* Session  
*from* schemas *import* schemas  
*from* services *import* generalServices  
*from* models *import* models  
\_model = models.OrderBook  
  
  
  
*def* add\_to\_basket(*db*: Session, *model*: schemas.OrderBookCreate, *current\_user*: models.User) -> int:  
 generalServices.check\_in\_warehouse(db=*db*, model=models.Book, id=*model*.book\_id, count=*model*.count)  
 book = generalServices.get\_by\_expression(db=*db*, model=models.Book, expression=models.Book.id == *model*.book\_id)  
 order\_book = *db*.query(\_model).filter(and\_(\_model.order\_id == 0, \_model.book\_id == *model*.book\_id)).first()  
 book.count -= *model*.count  
 *if* order\_book:  
 order\_book.count += *model*.count  
 *db*.commit()  
 *return* order\_book.id  
 *else*:  
 \_book = \_model(  
 order\_id=0,  
 consumer=*current\_user*,  
 book\_id=book.id,  
 count=*model*.count  
 )  
 *db*.add(\_book)  
 *db*.commit()  
 *return* \_book.id  
  
  
*def* delete\_from\_basket(*db*: Session, *id*: int):  
 order\_book = generalServices.get\_by\_expression(db=*db*, model=\_model, expression=\_model.id == *id*)  
 book = generalServices.get\_by\_expression(db=*db*, model=models.Book, expression=models.Book.id == order\_book.book\_id)  
 book.count += order\_book.count  
 *db*.delete(order\_book)  
 *db*.commit()

***Schemas.py :***

*from* datetime *import* datetime  
*from* typing *import* List, Optional  
*from* pydantic *import* BaseModel  
  
  
# books  
  
*class* BookBase(BaseModel):  
 name: str  
 author: str  
 content: Optional[str] = *None* price: float  
 count: int  
 publication\_date: datetime  
 ISBN: str  
  
*class* BookCreate(BookBase):  
 *pass  
  
  
class* BookDto(BookBase):  
 id: int  
 owner\_id: int  
 image: str  
 update\_date: datetime  
  
 *class* Config:  
 orm\_mode = *True*# order book  
*class* OrderBookBase(BaseModel):  
 count: int  
 book\_id: int  
  
  
*class* OrderBookCreate(OrderBookBase):  
 *pass  
  
class* OrderBook(OrderBookBase):  
 id: int  
 order\_id: int  
 consumer\_id: int  
  
 *class* Config:  
 orm\_mode = *True*# users  
*class* UserBase(BaseModel):  
 email: str  
 username: str  
 first\_name: Optional[str] = *None* last\_name: Optional[str] = *None  
  
  
class* UserCreate(UserBase):  
 password: Optional[str] = *None  
  
  
class* UserDto(UserBase):  
 id: int  
 role\_id: int = *None  
  
 class* Config:  
 orm\_mode = *True  
  
  
class* User(UserDto):  
 books: List[BookDto] = []  
  
 *class* Config:  
 orm\_mode = *True*# roles  
*class* RoleBase(BaseModel):  
 name: str  
  
  
*class* RoleCreate(RoleBase):  
 *pass  
  
class* RoleDto(RoleBase):  
 id: int  
  
 *class* Config:  
 orm\_mode = *True  
  
class* Role(RoleDto):  
 users: List[UserDto] = []  
  
 *class* Config:  
 orm\_mode = *True*# token  
*class* Token(BaseModel):  
 access\_token: str  
 token\_type: str  
  
  
*class* TokenData(BaseModel):  
 email: Optional[str] = *None*# orders  
*class* OrderBase(BaseModel):  
 deliver\_date: datetime  
 country: str  
 city: str  
 address: str  
  
  
*class* OrderCreate(OrderBase):  
 *pass  
  
  
class* Order(OrderBase):  
 id: int  
 user\_id: int  
 books: List[OrderBook] = []  
  
 *class* Config:  
 orm\_mode = *True*# genres  
*class* GenreBase(BaseModel):  
 name: str  
  
  
*class* GenreCreate(GenreBase):  
 *pass  
  
  
class* GenreDto(GenreBase):  
 id: int  
  
 *class* Config:  
 orm\_mode = *True  
  
  
class* Genre(GenreDto):  
 books: List[BookDto] = []  
  
 *class* Config:  
 orm\_mode = *True  
  
  
class* Book(BookDto):  
 orders: List[OrderBook] = []

genres: List[GenreDto] = []  
  
 *class* Config:  
 orm\_mode = *True*

***Back-end routes:***

*Catalog routes:*

*basket.py :*

*from* fastapi *import* APIRouter, Depends  
*from* typing *import* List  
*from* models *import* models  
*from* schemas *import* schemas  
*from* services *import* generalServices, basketService  
*from* config.dependencies *import* get\_db, get\_current\_user  
*from* config.db *import* Session  
*from* starlette *import* status  
*from* Exceptions *import* CustomAccessForbiddenException  
*from* sqlalchemy *import* and\_  
  
router = APIRouter(prefix='/basket', tags=['basket'])  
\_model = models.OrderBook  
\_admin\_role\_name = 'Admin'  
  
  
  
  
@router.get('', response\_model=List[schemas.OrderBook])  
*async def* basket(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 expression = and\_(\_model.order\_id == 0, \_model.consumer == *current\_user*)  
 *return* generalServices.get\_all\_with\_expression(db=*db*, model=\_model, skip=*skip*, limit=*limit*, expression=expression)  
  
  
@router.post('', response\_model=int)  
*async def* add\_to\_basket(*orderBookCreate*: schemas.OrderBookCreate, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 *return* basketService.add\_to\_basket(db=*db*, model=*orderBookCreate*, current\_user=*current\_user*)  
  
@router.delete('/{id:int}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* delete\_from\_basket(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* basketService.delete\_from\_basket(db=*db*, id=*id*)

*books.py :*

*from* typing *import* List, Optional  
  
*from* fastapi *import* APIRouter, Depends, UploadFile, File  
*from* starlette *import* status  
  
*from* config.db *import* Session  
*from* models *import* models  
*from* schemas *import* schemas  
*from* services *import* generalServices, bookServices  
*from* config.dependencies *import* get\_db, get\_current\_user  
*from* Exceptions *import* CustomAccessForbiddenException  
  
router = APIRouter(prefix='/books', tags=['books'])  
\_admin\_role\_name = 'Admin'  
\_model = models.Book  
  
  
@router.get('/unsorted', response\_model=List[schemas.BookDto])  
*async def* get\_books(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db)):  
 *return* generalServices.get\_all(db=*db*, model=\_model, skip=*skip*, limit=*limit*)  
  
  
@router.get('', response\_model=List[schemas.BookDto])  
*async def* get\_books(*query*: str = 'name', *reverse*: bool = *False*, *skip*: int=0, *limit*: int=10, *genre*: str = 'all',  
 *db*: Session = Depends(get\_db)):  
 *return* bookServices.get\_sorted\_books(db=*db*, skip=*skip*, limit=*limit*, query=*query*, reverse=*reverse*, genre=*genre*)  
  
  
@router.get('/search', response\_model=List[schemas.BookDto])  
*async def* get\_books(*searchQuery*: str, *query*: str = 'name', *reverse*: bool = *False*, *skip*: int=0, *genre*: str = 'all',  
 *limit*: int=10, *db*: Session = Depends(get\_db)):  
 expression = \_model.name.contains(*searchQuery*.lower())  
 *return* bookServices.get\_sorted\_books(db=*db*, skip=*skip*, limit=*limit*, query=*query*, reverse=*reverse*, genre=*genre*, filter\_expression=expression)  
  
  
@router.get('/my', response\_model=List[schemas.BookDto])  
*async def* get\_books(*query*: str = 'name', *reverse*: bool = *False*, *skip*: int=0, *limit*: int=10, *genre*: str = 'all',  
 *db*: Session = Depends(get\_db), *current\_user*: models.User = Depends(get\_current\_user)):  
 expression = \_model.owner == *current\_user  
 if current\_user*.role.name == \_admin\_role\_name:  
 expression = *True  
 return* bookServices.get\_sorted\_books(db=*db*, skip=*skip*, limit=*limit*, query=*query*, reverse=*reverse*, genre=*genre*, filter\_expression=expression)  
  
  
@router.get('/{id:int}', response\_model=schemas.Book)  
*async def* get\_book(*id*: int, *db*: Session = Depends(get\_db)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
@router.get('/{id:int}/dto', response\_model=schemas.BookDto)  
*async def* get\_book\_dto(*id*: int, *db*: Session = Depends(get\_db)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.get("/{id:int}/image", status\_code=status.HTTP\_200\_OK)  
*async def* get\_image(*id*: int, *db*: Session = Depends(get\_db)):  
 *return* bookServices.get\_image(db=*db*, id=*id*)  
  
  
@router.get("/image", status\_code=status.HTTP\_200\_OK)  
*async def* get\_image(*name*: str):  
 *return* bookServices.get\_image\_by\_name(*name*)

@router.post('', response\_model=int)  
*async def* create\_book(*bookCreate*: schemas.BookCreate,*db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 *return* bookServices.create\_book(db=*db*, model=*bookCreate*, current\_user=*current\_user*)  
  
  
@router.put('/{id:int}/update', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* update\_book(*id*: int, *bookUpdate*: schemas.BookCreate,  
 *db*: Session = Depends(get\_db), current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id* bookServices.update\_book(db=*db*, model=*bookUpdate*, expression=expression)  
  
  
@router.patch('/{id:int}/set\_genres', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* set\_genres(*id*: int, *genres*: List[schemas.GenreBase], *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id* bookServices.set\_genres(db=*db*, genres=*genres*, expression=expression)  
  
  
@router.patch('/{id:int}/change\_image', response\_model=str)  
*async def* change\_image(*id*: int, *db*: Session = Depends(get\_db), *file*: UploadFile = File(...),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id  
 return* bookServices.change\_image(db=*db*, image=*file*, expression=expression)  
  
  
@router.delete('/{id}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* delete\_book(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
  
 bookServices.delete\_book(db=*db*, id=*id*)

*genres.py :*

*from* typing *import* List  
  
*from* fastapi *import* APIRouter, Depends  
*from* starlette *import* status  
  
*from* Exceptions *import* CustomAccessForbiddenException  
*from* config.db *import* Session  
*from* config.decorators *import* raise\_403\_if\_not\_admin  
*from* models *import* models  
*from* schemas *import* schemas  
*from* services *import* generalServices, genreServices  
*from* config.dependencies *import* get\_db, get\_current\_user  
  
router = APIRouter(prefix='/genres', tags=['genres'])  
\_model = models.Genre  
  
  
@router.get('', response\_model=List[schemas.GenreDto])  
*async def* get\_genres(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db)):  
 *return* generalServices.get\_all(db=*db*, model=\_model, skip=*skip*, limit=*limit*)  
  
@router.get('/all', response\_model=List[schemas.GenreDto])  
*async def* get\_genres(*db*: Session = Depends(get\_db)):  
 *return* generalServices.get\_all\_without\_limit(db=*db*, model=\_model, expression=*True*)  
  
@router.get('/{id:int}', response\_model=schemas.Genre)  
*async def* get\_genre\_by\_id(*id*: int, *db*: Session = Depends(get\_db)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.get('/name', response\_model=schemas.Genre)  
*async def* get\_genre\_by\_name(*name*: str, *db*: Session = Depends(get\_db)):  
 expression = \_model.name == *name  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.post('', response\_model=int)  
@raise\_403\_if\_not\_admin  
*async def* create\_genre(*genreCreate*: schemas.GenreCreate, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* genreServices.create\_genre(db=*db*, model=*genreCreate*)  
  
  
@router.put('/{id:int}/update', status\_code=status.HTTP\_204\_NO\_CONTENT)  
@raise\_403\_if\_not\_admin  
*async def* update\_genre(*id*: int, *genreUpdate*: schemas.GenreCreate, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id* genreServices.update\_genre(db=*db*, model=*genreUpdate*, expression=expression)  
  
  
@router.delete('/{id}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
@raise\_403\_if\_not\_admin  
*async def* delete\_genre(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 generalServices.delete(db=*db*, model=\_model, id=*id*)

*orders.py :*

*from* typing *import* List  
  
*from* fastapi *import* APIRouter, Depends  
*from* starlette *import* status  
  
*from* Exceptions *import* CustomAccessForbiddenException  
*from* config.db *import* Session  
*from* config.decorators *import* raise\_403\_if\_not\_admin  
*from* models *import* models  
*from* schemas *import* schemas  
*from* services *import* generalServices, genreServices, orderServices  
*from* config.dependencies *import* get\_db, get\_current\_user  
  
router = APIRouter(prefix='/orders', tags=['orders'])  
\_model = models.Order  
\_admin\_role\_name = 'Admin'  
  
  
@router.get('', response\_model=List[schemas.Order])  
@raise\_403\_if\_not\_admin  
*async def* get\_orders(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* generalServices.get\_all(db=*db*, model=\_model, skip=*skip*, limit=*limit*)  
  
  
@router.get('/my', response\_model=List[schemas.Order])  
*async def* get\_orders(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 expression = \_model.user\_id == *current\_user*.id  
 *return* generalServices.get\_all\_with\_expression(db=*db*, model=\_model, skip=*skip*, limit=*limit*, expression=expression)  
  
  
@router.get('/{id:int}', response\_model=schemas.Order)  
*async def* get\_order(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.post('', response\_model=int)  
*async def* create\_order(*orderCreate*: schemas.OrderCreate, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 *return* orderServices.create\_order(db=*db*, model=*orderCreate*, current\_user=*current\_user*)  
  
  
@router.delete('/{id:int}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* delete\_order(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 orderServices.delete\_order(db=*db*, id=*id*)

*User routes :*

*roles.py :*

*from* typing *import* List  
  
*from* fastapi *import* APIRouter, Depends  
*from* starlette *import* status  
  
*from* config.db *import* Session  
*from* config.decorators *import* raise\_403\_if\_not\_admin  
*from* models *import* models  
*from* schemas *import* schemas

*from* services *import* generalServices, roleServices  
*from* config.dependencies *import* get\_db, get\_current\_user  
*from* Exceptions *import* CustomAccessForbiddenException  
  
router = APIRouter(prefix='/roles', tags=['roles'])  
\_admin\_role\_name = 'Admin'  
\_model = models.Role  
  
@router.get('', response\_model=List[schemas.RoleDto])  
@raise\_403\_if\_not\_admin  
*async def* get\_roles(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* generalServices.get\_all(db=*db*, model=models.Role, skip=*skip*, limit=*limit*)  
  
  
@router.get('/{id:int}', response\_model=schemas.Role)  
@raise\_403\_if\_not\_admin  
*async def* get\_role\_by\_id(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.get('/name', response\_model=schemas.Role)  
@raise\_403\_if\_not\_admin  
*async def* get\_role\_by\_name(*name*: str, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.name == *name  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.post('', response\_model=int)  
@raise\_403\_if\_not\_admin  
*async def* create\_role(*roleCreate*: schemas.RoleCreate, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* roleServices.create\_role(db=*db*, model=*roleCreate*)  
  
  
@router.put('/{id:int}/update', status\_code=status.HTTP\_204\_NO\_CONTENT)  
@raise\_403\_if\_not\_admin  
*async def* update\_role\_by\_id(*id*: int, *roleUpdate*: schemas.RoleCreate, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id* roleServices.update\_role(db=*db*, model=*roleUpdate*, expression=expression)  
  
  
@router.delete('/{id:int}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
@raise\_403\_if\_not\_admin  
*async def* delete\_role\_by\_id(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 generalServices.delete(db=*db*, model=models.Role, id=*id*)

*users.py:*

*from* typing *import* List  
  
*from* fastapi *import* APIRouter, Depends  
*from* starlette *import* status  
  
*from* Exceptions *import* CustomAccessForbiddenException  
*from* config.db *import* Session  
*from* config.decorators *import* raise\_403\_if\_not\_admin  
*from* models *import* models  
*from* schemas *import* schemas  
*from* services *import* generalServices, userServices  
*from* config.dependencies *import* get\_db, get\_current\_user  
\_admin\_role\_name = 'Admin'  
  
router = APIRouter(prefix='/users',tags=['users'])  
\_model = models.User  
  
@router.get('', response\_model=List[schemas.UserDto])  
@raise\_403\_if\_not\_admin  
*async def* get\_users(*skip*: int=0, *limit*: int=10, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 *return* generalServices.get\_all(db=*db*, model=\_model, skip=*skip*, limit=*limit*)  
  
  
@router.get('/{id:int}', response\_model=schemas.User)  
@raise\_403\_if\_not\_admin  
*async def* get\_user\_by\_id(*id*: int, *db*: Session = Depends(get\_db), current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
  
@router.get('/email', response\_model=schemas.User)  
@raise\_403\_if\_not\_admin  
*async def* get\_user\_by\_email(*email*: str, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 expression = \_model.email == *email  
 return* generalServices.get\_by\_expression(db=*db*, model=\_model, expression=expression)  
  
@router.get("/me", response\_model=schemas.UserDto)  
*async def* get\_current\_user(*current\_user*: models.User = Depends(get\_current\_user)):  
 *return current\_user*@router.post('', response\_model=int)  
*async def* create\_user(*userCreate*: schemas.UserCreate, *db*: Session = Depends(get\_db)):  
 *return* userServices.create\_user(db=*db*, model=*userCreate*)  
  
  
@router.put('/{id:int}/update', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* update\_user\_by\_id(*id*: int, *userUpdate*: schemas.UserCreate, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 expression = \_model.id == *id  
 if current\_user*.role.name == \_admin\_role\_name:  
 *return* userServices.update\_user(db=*db*, model=*userUpdate*, expression=expression)  
  
 *if not current\_user*.id == *id*: CustomAccessForbiddenException()  
 *else*:  
 userServices.update\_user(db=*db*, model=*userUpdate*, expression=expression)  
  
  
@router.put('/email/update', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* update\_user\_by\_email(*email*: str, *userUpdate*: schemas.UserCreate, *db*: Session = Depends(get\_db),  
 *current\_user*: models.User = Depends(get\_current\_user)):  
 expression = \_model.email == *email  
 if current\_user*.role.name == \_admin\_role\_name:  
 *return* userServices.update\_user(db=*db*, model=*userUpdate*, expression=expression)  
  
 *if not current\_user*.email == *email*: CustomAccessForbiddenException()  
 *else*:  
 userServices.update\_user(db=*db*, model=*userUpdate*, expression=expression)  
  
  
@router.patch('/{id:int}/change\_role', status\_code=status.HTTP\_204\_NO\_CONTENT)  
*async def* change\_role(*id*: int, *role\_name*: str, *db*: Session = Depends(get\_db)):  
 userServices.change\_user\_role(db=*db*, user\_id=*id*, role\_name=*role\_name*)  
  
  
@router.delete('/{id:int}/delete', status\_code=status.HTTP\_204\_NO\_CONTENT)  
@raise\_403\_if\_not\_admin  
*async def* delete\_user\_by\_id(*id*: int, *db*: Session = Depends(get\_db),  
 current\_user: models.User = Depends(get\_current\_user)):  
 userServices.delete\_user(db=*db*, id=*id*)

*security.py :*

*from* datetime *import* timedelta  
  
*from* fastapi *import* Depends, APIRouter, HTTPException  
*from* fastapi.security *import* OAuth2PasswordRequestForm  
*from* starlette *import* status  
  
*from* config *import* ACCESS\_TOKEN\_EXPIRE\_MINUTES  
*from* config.db *import* Session  
*from* config.dependencies *import* get\_db  
*from* schemas *import* schemas  
*from* services.securityServices *import* authenticate\_user, create\_access\_token  
  
router = APIRouter(tags=['security'])  
  
@router.post("/token", response\_model=schemas.Token)  
*async def* login\_for\_access\_token(*form\_data*: OAuth2PasswordRequestForm = Depends(), *db*: Session = Depends(get\_db)):  
 user = authenticate\_user(db=*db*, email=*form\_data*.username, password=*form\_data*.password)  
 *if not* user:  
 *raise* HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.email, "role": user.role.name}, expires\_delta=access\_token\_expires  
 )  
 *return* {"access\_token": access\_token, "token\_type": "bearer"}

***Back-end models.py :***

*from* datetime *import* datetime  
  
*from* sqlalchemy *import* Float, Column, ForeignKey, Integer, String, DateTime, Table  
*from* sqlalchemy.orm *import* relationship, backref  
  
*from* config.db *import* Base  
  
*class* User(Base):  
 \_\_tablename\_\_ = 'users'  
  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 email = Column(String, unique=*True*, index=*True*, nullable=*False*)  
 password = Column(String, nullable=*False*)  
 username = Column(String, index=*True*, nullable=*False*)  
 first\_name = Column(String)  
 last\_name = Column(String)  
  
 role\_id = Column(Integer, ForeignKey('roles.id'))  
 role = relationship('Role', back\_populates='users')  
 books = relationship('Book', back\_populates='owner')  
 basket = relationship('OrderBook', back\_populates='consumer')  
  
*class* Role(Base):  
 \_\_tablename\_\_ = 'roles'  
  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 name = Column(String, unique=*True*, index=*True*, nullable=*False*)  
  
 users = relationship('User', back\_populates='role')  
  
books\_genres\_temporary\_table = \  
 Table('books\_genres', Base.metadata,  
 Column('book\_id', ForeignKey('books.id'), primary\_key=*True*),  
 Column('genre\_id', ForeignKey('genres.id'), primary\_key=*True*)  
 )  
  
*class* Book(Base):  
 \_\_tablename\_\_ = 'books'  
  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 name = Column(String, nullable=*False*)  
 author = Column(String, nullable=*False*)  
 content = Column(String)  
 price = Column(Float, nullable=*False*)  
 publication\_date = Column(DateTime, default=datetime.utcnow)  
 update\_date = Column(DateTime, default=datetime.utcnow)  
 count = Column(Integer)  
 image = Column(String, default='default.png')  
 ISBN = Column(String, unique=*True*,index=*True*, nullable=*False*)  
  
 owner\_id = Column(Integer, ForeignKey('users.id'))  
 owner = relationship('User', back\_populates='books')  
 orders = relationship('OrderBook', back\_populates='book')  
 genres = relationship('Genre', secondary=books\_genres\_temporary\_table, back\_populates='books', lazy=*True*)  
  
  
*class* Order(Base):  
 \_\_tablename\_\_ = 'orders'  
  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 deliver\_date = Column(DateTime)  
 country = Column(String)  
 city = Column(String)  
 address = Column(String)  
  
 user\_id = Column(Integer, ForeignKey('users.id'))  
 books = relationship('OrderBook', back\_populates='order')  
  
*class* OrderBook(Base):  
 \_\_tablename\_\_ = 'orders\_books'  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 count = Column(Integer)  
  
 consumer\_id = Column(Integer, ForeignKey('users.id'))  
 consumer = relationship('User', back\_populates='basket')  
 order\_id = Column(Integer, ForeignKey('orders.id'))  
 order = relationship('Order', back\_populates='books')  
 book\_id = Column(Integer, ForeignKey('books.id'))  
 book = relationship('Book', back\_populates='orders')  
  
  
*class* Genre(Base):  
 \_\_tablename\_\_ = 'genres'  
  
 id = Column(Integer, primary\_key=*True*, index=*True*)  
 name = Column(String, unique=*True*, index=*True*, nullable=*False*)  
  
 books = relationship('Book', secondary=books\_genres\_temporary\_table, back\_populates='genres', lazy=*True*)

***Back-end config :***

*db.py :*

*from* sqlalchemy *import* create\_engine  
*from* sqlalchemy.ext.declarative *import* declarative\_base  
*from* sqlalchemy.orm *import* sessionmaker  
  
SQLALCHEMY\_DATABASE\_URL = "sqlite:///./database.config"  
  
engine = create\_engine(  
 SQLALCHEMY\_DATABASE\_URL, connect\_args={"check\_same\_thread": *False*}  
)  
Session = sessionmaker(autocommit=*False*, autoflush=*False*, bind=engine)  
  
Base = declarative\_base()

*decorators.py :*

*from* functools *import* wraps  
*from* Exceptions *import* CustomAccessForbiddenException  
*def* raise\_403\_if\_not\_admin(*func*):  
 @wraps(func)  
 *async def* wrapper(*\*args*, *\*\*kwargs*):  
 *if* (user := *kwargs*.get('current\_user')) *and* user.role.name != 'Admin':  
 CustomAccessForbiddenException()  
 *return await* func(\**args*, \*\**kwargs*)  
  
 *return* wrapper

*dependencies.py :*

*from* fastapi *import* Depends, HTTPException  
*from* jose *import* JWTError, jwt  
*from* starlette *import* status  
  
*from* models *import* models  
*from* schemas.schemas *import* TokenData  
*from* . *import* SECRET\_KEY, ALGORITHM  
*from* .db *import* Session, Base, engine  
*from* fastapi.security *import* OAuth2PasswordBearer  
*from* services *import* generalServices, securityServices  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="api/token")  
  
*def* create\_database():  
 Base.metadata.create\_all(bind=engine)  
 db: Session = Session()  
 roles = db.query(entity:=models.Role).all()  
 *if not* roles:  
 admin\_role = entity(name='Admin')  
 user\_role = entity(name='User')  
 db.add\_all([admin\_role, user\_role])  
 db.commit()  
  
 *if* 'Admin' *not in* [user.role.name *for* user *in* db.query(models.User).all()]:  
 admin = models.User(  
 email='admin@bookstore.com',  
 password=securityServices.get\_password\_hash('admin'),  
 username='Admin',  
 role\_id=1  
 )  
 db.add(admin)  
 db.commit()  
  
  
  
  
*def* get\_db():  
 db = Session()  
 *try*:  
 *yield* db  
 *finally*:  
 db.close()  
  
  
*async def* get\_current\_user(*token*: str = Depends(oauth2\_scheme), *db*: Session = Depends(get\_db)):  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 *try*:  
 payload = jwt.decode(*token*, SECRET\_KEY, algorithms=[ALGORITHM])  
 email: str = payload.get("sub")  
 *if* email *is None*:  
 *raise* credentials\_exception  
 token\_data = TokenData(email=email)  
 *except* JWTError:  
 *raise* credentials\_exception  
  
 model = models.User  
 expression = model.email == token\_data.email  
 user = generalServices.get\_by\_expression(db=*db*, model=model, expression=expression)  
  
 *if* user *is None*:  
 *raise* credentials\_exception  
 *return* user

**Front-end:**

***Main.js :***

import { *createApp*} from 'vue'  
import App from '@/App.vue'  
import components from '@/components/UI'  
import router from "@/router/router"  
import directives from "@/directives/directives"  
import store from '@/store'  
import 'bootstrap/dist/css/bootstrap.css'  
import 'bootstrap-vue/dist/bootstrap-vue.css'  
  
import VueSweetalert2 from 'vue-sweetalert2';  
import 'sweetalert2/dist/sweetalert2.min.css';  
  
const app = *createApp*(App)  
  
components.forEach(*component* =>{  
 app.component(*component*.name, *component*)  
})  
  
directives.forEach(*directive* =>  
 app.directive(*directive*.name, *directive*))  
  
const options = {  
 confirmButtonColor: '#7072F7',  
 cancelButtonColor: '#ff7674',  
 background: 'rgba(56, 58, 73)',  
  
};  
  
app  
 .use(router)  
 .use(store)  
 .use(VueSweetalert2, options)  
 .mount('#app')

***App.vue :***

<template>  
 <div class**="body"**>  
 <navbar class**="position-fixed w-100"**></navbar>  
 <div class**="app"**>  
 <router-view></router-view>  
 </div>  
 </div>  
</template>  
  
<script>  
export default {  
 mounted() {  
  
 if (*Date*.now() >= *localStorage*.getItem('tokenExp') \* 1000) {  
 this.$store.dispatch('logout')  
 this.$swal({  
 title: "<span style='color: #ffffff'>Oops...</span>",  
 icon: 'warning',  
 html: "<span style='color: #ffffff'>Token expired</span>"  
 })  
 }  
 else if(*JSON*.parse(*localStorage*.getItem('isAuth'))){  
 this.$store.commit('setToken', *localStorage*.getItem('accessToken'))  
 this.$store.commit('setAuth', *JSON*.parse(*localStorage*.getItem('isAuth')))  
 this.$store.commit('setAdmin', *JSON*.parse(*localStorage*.getItem('isAdmin')))  
 this.$store.commit('setExp', *JSON*.parse(*localStorage*.getItem('tokenExp')))  
 }  
 },  
}  
</script>  
  
<style>  
html{  
 *background-color*: rgb(23, 28, 33);  
}  
  
\* {  
 *margin*: 0;  
 *padding*: 0;  
 *box-sizing*: **border-box**;  
 *font-family*: 'Merriweather', **serif**;  
 *font-weight*: 300;  
 *color*: **white**;  
}  
::-webkit-scrollbar-track  
{  
 *border-radius*: 50%;  
}  
  
::-webkit-scrollbar  
{  
 *width*: 0.5**em**;  
 *height*: 0.5**em**;  
}  
  
::-webkit-scrollbar-thumb  
{  
 *border-radius*: 10**px**;  
 *background-color*: rgba(56, 58, 73, 0.9);  
}

.app{  
 *padding*: 20**px**;  
 *padding-top*: 72**px**;  
}  
  
.body{  
 *min-height*: 100**vh**;  
 *min-width*: **fit-content**;  
 *background-color*: rgb(23, 28, 33);  
 *-moz-osx-font-smoothing*: **grayscale**;  
 *-webkit-text-size-adjust*: 100%;  
}  
  
.\_table {  
 *border-collapse*: **collapse**;  
 *width*: 100%;  
 *min-width*: 800**px**;  
}  
  
  
.form{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *background-color*: **inherit**;  
}  
  
.input {  
 *padding*: 10**px** 15**px**;  
 *margin-top*: 10**px**;  
 *margin-bottom*: 10**px**;  
 *color*: #292a5c;  
}  
  
.user\_\_page{  
 *display*: **flex**;  
 *justify-content*: **center**;  
}  
  
.user\_\_form{  
 *max-width*: 400**px**;  
 *width*: 85**vw**;  
}  
  
.observer{  
 *height*: 30**px**;  
}  
  
.app\_\_btns{  
 *display*: **flex**;  
 *justify-content*:**space-between**;  
 *gap*: 0.3**rem**}  
.link{  
 *color*: #7072F7;  
 *text-decoration*: **none**;  
}  
  
.error{  
 *color*: #FFF587;  
}

.list-item{

*display*: **inline-block**;  
 *margin-right*: 10**px**;  
}  
.list-enter-active,  
.list-leave-active{  
 *transition*: **all** 0.7**s ease**;  
}  
  
.list-enter-from,  
.list-leave-to{  
 *opacity*: 0;  
 *transform*: translateX(130**px**);  
}  
  
.list-move{  
 *transition*: transform 0.4**s ease**;  
}  
  
  
  
</style>

***Front-end store :***

*basketModule.js:*

import {instance} from "@/router/instance";  
  
  
export const *basketModule* = {  
 state: () => ({  
 books: [],  
 visible: false,  
 isLoading: false,  
 page: 0,  
 limit: 10,  
 defaultRoot: 'catalog/basket',  
 order: {count: 1},  
 isAll: false  
 }),  
 getters: {  
 getTotalSum(*state*){  
 if(*state*.books.length > 0)  
 return ([...*state*.books]  
 .map(*book* => *book*.price \* *book*.order\_count))  
 .reduce((*previous*, *next*) => *previous* + *next*)  
 }  
 },  
 mutations: {  
 setBooks(*state*, *books*){  
 *state*.books = *books*;  
 },  
 addBooks(*state*, *books*){  
 *state*.books = [...*state*.books, ...*books*];  
 },  
 pushBook(*state*, *book*){  
 *state*.books.push(*book*);  
 },  
 setVisibility(*state*, *bool*){  
 if(!*bool*) {  
 *state*.books = []

*state*.page = 0  
 }  
 *state*.visible = *bool* },  
 clearBasket(*state*) {  
 *state*.books = []  
 *state*.isAll = false  
 *state*.page = 0  
 },  
 setLoading(*state*, *bool*){  
 *state*.isLoading = *bool* },  
 setAll(*state*, *bool*){  
 *state*.isAll = *bool* },  
  
  
 },  
 actions: {  
 async addToBasket({*state*, *commit*, *rootState*, *rootGetters*}, *book*){  
 *commit*('setLoading', true)  
 const order = {  
 count: *state*.order.count,  
 book\_id: *book*.id  
 }  
 if(*book*.count > 0) {  
 const count = {...*book*}.count - order.count  
 await instance  
 .post(*state*.defaultRoot, order, {headers: *rootGetters*.getHeaders})  
 .catch(*error* => {  
 *console*.log(*error*)  
 *rootState*.errors.push(*error*)  
 })  
 .then(() => {  
 if(*rootState*.errors.length === 0)  
 *rootState*.book.book.count = count  
 })  
 }  
 *commit*('setLoading', false)  
 },  
 async getBasketBooks({*state*, *commit*, *dispatch*, *rootState*, *rootGetters*}, *isAdmin*=false){  
 if(!*state*.isAll) {  
 if(*state*.books.length === 0) {  
 *state*.page = 0  
 await *commit*('setLoading', true)  
 }  
 *rootState*.errors = []  
 *state*.page += 1  
 let params = {  
 skip: (*state*.page - 1) \* *state*.limit,  
 limit: *state*.limit  
 }  
 const config = {params: params}  
 config.headers = *rootGetters*.getHeaders  
 await instance  
 .get(*state*.defaultRoot, config)  
 .then(*response* => {  
 if(*response*.data.length === 0 && *state*.page !== 1)  
 *commit*('setAll', true)  
 *response*.data.forEach(async *orderBook* => {  
 const book = await *dispatch*('getBook', *orderBook*.book\_id)

book.id = *orderBook*.id  
 book.book\_id = *orderBook*.book\_id  
 book.order\_count = *orderBook*.count  
 await *commit*('pushBook', book)  
 })  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setLoading', false)  
 }  
 },  
 async getBook({*state*, *commit*, *rootState*}, *book\_id*){  
 const path = `catalog/books/${*book\_id*}/dto`  
 *rootState*.errors = []  
  
 return await instance  
 .get(path)  
 .then(*res* => *res*.data)  
 .catch(*error* => {  
 *console*.log(*error*)  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
 async removeFromBasket({*state*, *commit*, *rootState*, *rootGetters*}, *obj*){  
 const path = `${*state*.defaultRoot}/${*obj*.id}/delete`  
 await instance.delete(path, {headers: *rootGetters*.getHeaders})  
 .then(() => {  
 *rootState*.book.book.count = +*rootState*.book.book.count + +*obj*.count  
 *commit*('setBooks', *state*.books.filter(*x* => *x*.id !== *obj*.id ))  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 }  
  
 },  
 namespaced: true  
  
}

*bookModule.js :*

import {instance} from "@/router/instance";  
import router from "@/router/router";  
  
  
export const *bookModule* = {  
 state: () => ({  
 books: [],  
 book: {},  
 isLoading: false,  
 isFormLoading: false,  
 selectedSort: 'name',  
 reverseSort: {  
 value: false  
 },  
 searchQuery: {  
 value: "",  
 searched: false  
 },  
 page: 0,  
 limit: 10,  
 defaultRoot: 'catalog/books',  
 sortOptions: [  
 {value: 'name', name: 'By name'},  
 {value: 'author', name: 'By author'},  
 {value: 'price', name: 'By price'},  
 {value: 'count', name: 'By count'},  
 {value: 'publication\_date', name: 'By date'},  
 ],  
 urlCreator: *window*.*URL* || *window*.webkitURL,  
 isAll: false  
 }),  
 getters: {  
 sortedBooks(*state*){  
 return *state*.books  
 },  
 },  
 mutations: {  
 setBooks(*state*, *books*){  
 *state*.books = *books*;  
 },  
 addBooks(*state*, *books*){  
 *state*.books = [...*state*.books, ...*books*];  
 },  
 setBook(*state*, *book*){  
 *state*.book = *book* },  
 pushBook(*state*, *book*){  
 *state*.books.push(*book*);  
 },  
 setLoading(*state*, *bool*){  
 *state*.isLoading = *bool*;  
 },  
 setFormLoading(*state*, *bool*){  
 *state*.isFormLoading = *bool*;  
 },  
 setSelectedSort(*state*, *selectedSort*){  
 *state*.selectedSort = *selectedSort*;  
 },  
 setSearchQuery(*state*, *searchQuery*){  
 *state*.searchQuery = *searchQuery*;  
 },  
 setDefaultRoot(*state*, *defaultRoot*){  
 *state*.defaultRoot = *defaultRoot* },  
 setSortedBooks(*state*, *sortedBooks*){  
 *state*.sortedEvents = *sortedBooks* },  
 clearBookStore(*state*){  
 *state*.books = []  
 *state*.page = 0  
 *state*.searchQuery.searched = false  
 *state*.isAll = false  
 },  
 clearBooks(*state*){  
 *state*.books = []  
 *state*.page = 0  
 *state*.isAll = false  
 },

clearBook(*state*) {  
 *state*.book = {}  
 },  
 setSearchedMod(*state*, *bool*){

*state*.searchQuery.searched = *bool* },  
 setImage(*state*, *image*){  
 *state*.book.image = *image* },  
 setGenres(*state*, *genres*){  
 *state*.book.genres = *genres* },  
 setReverseSort(*state*, *bool*){  
 *state*.reverseSort = *bool* },  
 setBookCount(*state*, *count*){  
 *state*.book.count = *count* },  
 setAll(*state*, *bool*){  
 *state*.isAll = *bool* },  
  
 },  
 actions: {  
 async createBook({*state*, *commit*, *rootState*, *rootGetters*}) {  
 await *commit*('setFormLoading', true)  
 *rootState*.errors = []  
 const book = *state*.book  
 book.publication\_date = new *Date*(book.publication\_date.toLocaleString()).toISOString()  
 await instance  
 .post(`${*state*.defaultRoot}`, book, {headers: *rootGetters*.getHeaders})  
 .then(*response* => {  
 book.id = *response*.data  
 book.image = 'default.png'  
 *commit*('pushBook', book)  
 *commit*('clearBook')  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setFormLoading', false)  
 },  
 async getSortedBookList({*state*, *commit*, *rootState*, *rootGetters*}, *owner*) {  
 if(!*state*.isAll) {  
 if (*state*.books.length === 0) {  
 *commit*('setLoading', true)  
 }  
 *state*.page += 1  
  
 let path = `${*state*.defaultRoot}`  
 let params = {  
 skip: (*state*.page - 1) \* *state*.limit,  
 limit: *state*.limit,  
 query: *state*.selectedSort,  
 reverse: *state*.reverseSort.value,  
 genre: *rootState*.genre.selectedSort  
 }  
 *console*.log(params)  
  
 if (*state*.searchQuery.searched) {

path += '/search'  
 params.searchQuery = *state*.searchQuery.value  
 }  
  
 const config = {params: params}

if (*owner* === true) {  
 path += '/my'  
 config.headers = *rootGetters*.getHeaders  
 }  
  
 await instance  
 .get(path, config)  
 .then(async *response* => {  
 if(*response*.data.length === 0)  
 *commit*('setAll', true)  
 await *response*.data.forEach(async *book* => {  
 if (*state*.books.filter(*b* => *b*.id === *book*.id).length === 0)  
 *commit*('pushBook', *book*)  
 })  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setLoading', false)  
 }  
 },  
  
 async getBook({*state*, *commit*, *rootState*}, *book\_id*){  
 const path = `${*state*.defaultRoot}/${*book\_id*}`  
 *rootState*.errors = []  
  
 await instance  
 .get(path)  
 .then(*response* => {  
 *response*.data.publication\_date = new *Date*(*response*.data.publication\_date).toISOString().split('T')[0]  
 *response*.data.update\_date = new *Date*(*response*.data.update\_date).toISOString().split('T')[0]  
 *commit*('setBook', *response*.data)  
 })  
 .catch(*error* => {  
 if (*error*.response.status === 404)  
 router.push({name: 'NotFound'})  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
 async updateBook({*state*, *commit*, *rootState*, *rootGetters*}) {  
 await *commit*('setFormLoading', true)  
 *rootState*.errors = []  
 const book = {...*state*.book}  
 book.publication\_date = new *Date*(book.publication\_date.toLocaleString()).toISOString()  
 const path = `${*state*.defaultRoot}/${book.id}/update`  
 await instance  
 .put(path, book, {headers: *rootGetters*.getHeaders})  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setFormLoading', false)  
 },  
 async removeBook({*state*, *commit*, *rootState*, *rootGetters*}, *book\_id*){  
 const path = `${*state*.defaultRoot}/${*book\_id*}/delete`

await instance.delete(path, {headers: *rootGetters*.getHeaders})  
 .then(() => {  
 *commit*('setBooks', *state*.books.filter(*x* => *x*.id !== *book\_id* ))  
 })  
 .catch(*error* => {

*rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
 async getBookImage({*state*, *commit*, *rootState*}, *obj*){  
 const path = `${*state*.defaultRoot}/image`  
 const image\_name = *obj*.image\_name  
 const target = *obj*.target  
 if(*obj*.image\_name !== undefined) {  
 await instance  
 .get(path, {  
 responseType: 'blob',  
 params: {name: image\_name}  
 })  
 .then(*response* => {  
 const blob = new *Blob*(  
 [*response*.data],  
 {  
 type: *response*.headers['content-type']  
 })  
 const url = *state*.urlCreator.createObjectURL(blob)  
 target.setAttribute('src', url);  
 })  
 .catch(*error* => {  
 *console*.log(*error*)  
 *rootState*.errors.push(*error*)  
 })  
 }  
 },  
 async changeBookImage({*state*, *commit*, *rootState*}){  
 await *commit*('setLoading', true)  
 *rootState*.errors = []  
 const form = new *FormData*(*document*.querySelector('#uploadForm'))  
 const path = `${*state*.defaultRoot}/${*state*.book.id}/change\_image`  
  
 await instance  
 .patch(path, form, {  
 headers: {  
 'Content-Type': 'multipart/form-data',  
 Authorization: `Bearer ${*rootState*.accessToken}`,  
 }})  
 .then(*response* => {  
 *commit*('setImage', *response*.data)  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setLoading', false)  
 },  
 async changeGenres({*state*, *commit*, *rootGetters*}, *genres*){  
 await *commit*('setLoading', true)  
 const path = `${*state*.defaultRoot}/${*state*.book.id}/set\_genres`  
  
 const entity = []  
 *genres*.forEach(*genre* => { entity.push({name: *genre*}) })  
 await instance  
 .patch(path, entity, {headers: *rootGetters*.getHeaders})  
 .then(() => *commit*('setGenres', entity))

.catch(*error* => {  
 *console*.log(*error*)  
 })  
 await *commit*('setLoading', false)  
 },  
  
 },

namespaced: true  
  
}

*genreModule.js :*

import {instance} from "@/router/instance";  
import router from "@/router/router";  
  
  
export const *genreModule* = {  
 state: () => ({  
 genres: [],  
 genre: {},  
 page: 0,  
 limit: 10,  
 defaultRoot: 'catalog/genres',  
 isAll: false,  
 isLoading: false,  
 selectedSort: 'all',  
 sortOptions: [  
 {value: 'all', name: 'All'},  
 ],  
 }),  
 mutations: {  
 setGenres(*state*, *genres*){  
 *state*.genres = *genres*;  
 },  
 addGenres(*state*, *genres*){  
 *state*.genres = [...*state*.genres, ...*genres*];  
 },  
 setGenre(*state*, *genre*){  
 *state*.genre = *genre*;  
 },  
 pushGenre(*state*, *genre*){  
 *state*.genres.push(*genre*);  
 },  
 setLoading(*state*, *bool*){  
 *state*.isLoading = *bool*;  
 },  
 setDefaultRoot(*state*, *defaultRoot*){  
 *state*.defaultRoot = *defaultRoot* },  
 clearGenre(*state*){  
 *state*.genre = {}  
 },  
 setAll(*state*, *bool*){  
 *state*.isAll = *bool* },  
 clearGenreStore(*state*){  
 *state*.genres = []  
 *state*.page = 0  
 *state*.isAll = false  
 },  
 setSortedOptions(*state*, *options*){  
 *state*.sortOptions = [...*state*.sortOptions,...*options*]  
 },  
 pushOption(*state*, *option*){  
 *state*.sortOptions.push(*option*)  
 },  
 setSelectedSort(*state*, *selectedSort*){  
 *state*.selectedSort = *selectedSort*;  
 },  
 },  
 actions: {  
 async getGenres({*state*, *commit*, *rootState*}, *isAdmin* = false){  
 if(!*state*.isAll) {  
 *rootState*.errors = []  
 *state*.page += 1  
 let params = {  
 skip: (*state*.page - 1) \* *state*.limit,  
 limit: *state*.limit  
 }  
 const config = {params: params}  
 await instance  
 .get(*state*.defaultRoot, config)  
 .then(*response* => {  
 if(*response*.data.length === 0)  
 *commit*('setAll', true)  
 *commit*('addGenres', *response*.data)  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 }  
 },  
 async getAllGenres({*state*, *commit*, *rootState*}){  
 *rootState*.errors = []  
 const path = `${*state*.defaultRoot}/all`  
 await instance  
 .get(path)  
 .then(*response* => {  
 *response*.data.forEach(*genre* => {  
 const option = {value: *genre*.name, name: *genre*.name}  
 *commit*('pushOption', option)  
 })  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
 async createGenre({*state*, *commit*, *rootState*, *rootGetters*}){  
 await *commit*('setLoading', true)  
 *rootState*.errors = []  
 await instance  
 .post(*state*.defaultRoot, *state*.genre, {headers: *rootGetters*.getHeaders})  
 .then(*response* => {  
 *state*.genre.id = *response*.data  
 *commit*('pushGenre', *state*.genre)  
 *commit*('clearGenre')  
 })  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 await *commit*('setLoading', false)  
 },  
 async updateGenre({*state*, *rootState*, *rootGetters*}){  
 *rootState*.errors = []

const path = `${*state*.defaultRoot}/${*state*.genre.id}/update`  
 await instance  
 .put(path, *state*.genre, {headers: *rootGetters*.getHeaders})  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
 async removeGenre({*state*, *commit*, *rootState*, *rootGetters*}, *genre\_id*){

const path = `${*state*.defaultRoot}/${*genre\_id*}/delete`  
 await instance  
 .delete(path, {headers: *rootGetters*.getHeaders})  
 .then(() =>  
 *commit*('setGenres',  
 [...*state*.genres]  
 .filter(*genre* => *genre*.id !== *genre\_id* )  
 ),  
 )  
 .catch(*error* => {  
 *rootState*.errors.push(*error*.response.data.detail)  
 })  
 },  
  
 },  
 namespaced: true  
  
}

*orderModule.js :*

import {instance} from "@/router/instance";  
  
  
export const *orderModule* = {  
 state: () => ({  
 orders: [],  
 order: {},  
 defaultRoot: 'catalog/orders',  
 isLoading: false,  
 page: 0,  
 limit: 10,  
 isAll: false  
 }),  
 getters: {  
  
 },  
 mutations: {  
 setOrders(*state*, *orders*){  
 *state*.orders = *orders*;  
 },  
 addOrders(*state*, *orders*){  
 *state*.orders = [...*state*.orders, ...*orders*];  
 },  
 pushOrder(*state*, *order*){  
 *state*.orders.push(*order*);  
 },  
 setVisibility(*state*, *bool*){  
 *state*.visible = *bool* },  
 setLoading(*state*, *bool*){  
 *state*.isLoading = *bool* },  
 clearOrder(*state*){  
 *state*.order = {}  
 },  
 setAll(*state*, *bool*){  
 *state*.isAll = *bool* },  
 clearOrderStore(*state*){  
 *state*.orders = []  
 *state*.isAll = false  
 *state*.page = 0  
 }  
  
 },  
 actions: {  
 async createOrder({*state*, *commit*, *rootState*, *rootGetters*}){  
 await *commit*('setLoading', true)  
 *state*.order.deliver\_date = new *Date*(*state*.order.deliver\_date.toLocaleString()).toISOString()  
 await instance  
 .post(state.defaultRoot, state.order, {headers: rootGetters.getHeaders})  
 .then(response => {  
 const order = {...state.order}  
 order.id = response.data  
 order.books = [...rootState.basket.books]  
 commit('pushOrder', order)  
 commit('clearOrder')  
 rootState.basket.books = []  
 })  
 .catch(error => {  
 console.log(error)  
 rootState.errors.push(error.response.data.detail)  
 })  
 await commit('setLoading', false)  
 },  
 async getOrders({state, commit, rootState, rootGetters}, isAdmin){  
 if(!state.isAll) {  
 await commit('setLoading', true)  
 let path = state.defaultRoot  
 state.page += 1  
 const params =  
 {  
 skip: (state.page - 1) \* state.limit,  
 limit: state.limit  
 }  
  
 if (!isAdmin)  
 path += '/my'  
 rootState.errors = []  
 await instance  
 .get(path, {  
 params: params,  
 headers: rootGetters.getHeaders  
 })  
 .then(response => {  
 if(response.data.length === 0)  
 commit('setAll', true)  
  
 response.data.forEach(order => {  
 if (state.orders.filter(o => o.id === order.id).length === 0) {  
 order.books.forEach(book => book.order\_count = book.count)  
 commit('pushOrder', order)  
 }

})  
 })  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 await commit('setLoading', false)  
 }  
 },  
 async removeOrder({state, commit, rootState, rootGetters}, order\_id){

const path = `${state.defaultRoot}/${order\_id}/delete`  
 await instance  
 .delete(path, {headers: rootGetters.getHeaders})  
 .then(() =>  
 commit('setOrders',  
 [...state.orders]  
 .filter(order => order.id !== order\_id )  
 ),  
 )  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 },  
 },  
 namespaced: true  
  
}

*userModule.js :*

import jwt\_decode from 'jwt-decode'  
import router from "@/router/router";  
import {instance} from "@/router/instance";  
import {Form} from "vee-validate";  
  
export const userModule = {  
 state: () => ({  
 users: [],  
 user: {},  
 defaultRoot: 'users',  
 isLoading: false,  
 page: 0,  
 limit: 10,  
 isAll: false  
 }),  
 mutations: {  
 setUsers(state, users){  
 state.users = users  
 },  
 clearUsers(state){  
 state.users = []  
 state.page = 0  
 state.isAll = false  
 },  
 clearUser(state){  
 state.user = {}  
 },  
 setUser(state, user){  
 state.user = user  
 },  
 pushUser(state, user){  
 state.users.push(user)  
 },  
 setDefaultRoot(state, defaultRoot) {  
 state.defaultRoot = defaultRoot  
 },  
 setDefaultUserRoot(state, defaultUserRoot) {  
 state.defaultUserRoot = defaultUserRoot  
 },  
 setLoading(state, bool) {  
 state.isLoading = bool  
 },  
 setAll(state, bool){  
 state.isAll = bool  
 }  
 },  
 actions: {  
 async register({state, commit, rootState}){  
 await commit('setLoading', true)  
 rootState.errors = []  
 await instance  
 .post(state.defaultRoot, state.user)  
 .then(response => {  
 state.user.id = response.data  
 commit('pushUser', state.user)  
 })  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 .then(() =>{  
 if(rootState.errors.length === 0)  
 router.push('/login')  
 })  
 await commit('setLoading', false)  
 },  
 async login({state, dispatch, rootState}){  
 const path = `/token`  
 const data = new FormData()  
 data.append('username', state.user.email)  
 data.append('password', state.user.password)  
 rootState.errors = []  
 await instance  
 .post(path, data)  
 .then(response =>{  
 rootState.accessToken = response.data.access\_token  
 rootState.isAuth = true  
 })  
 .catch(() => {  
 rootState.errors.push("Incorrect email or password")  
 })  
 .then(() => {  
 if (rootState.errors.length === 0) {  
 dispatch('decodeRoleFromJWT')  
 router.push('/')  
 }  
 })  
 },  
 async getUserById({state, commit, rootState, rootGetters}, user\_id){  
 const path = `${state.defaultRoot}/${user\_id}`  
 await instance  
 .get(path, {headers: rootGetters.getHeaders})  
 .then(response => {  
 commit('setUser', response.data)  
 })  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })

},  
 async getUserByEmail({state, rootState, rootGetters}, email){  
 const path = `${state.defaultRoot}/${email}/email`  
 return await instance  
 .get(path, {headers: rootGetters.getHeaders})  
 .then(response => response.data)  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 },

async getUsers({state, commit, rootState, rootGetters}){  
 if(!state.isAll) {  
 rootState.errors = []  
 state.page += 1  
 await instance  
 .get(state.defaultRoot, {  
 params: {  
 skip: (state.page - 1) \* state.limit,  
 limit: state.limit  
 },  
 headers: rootGetters.getHeaders  
 })  
 .then(response => {  
 if(response.data.length === 0)  
 commit('setAll', true)  
 response.data.forEach(user => {  
 user.role = user.role\_id === 1  
 commit('pushUser', user)  
 })  
 })  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 }  
 },  
 async removeUser({state, commit, rootState, rootGetters}, user\_id){  
 const path = `${state.defaultRoot}/${user\_id}/delete`  
 await instance  
 .delete(path, {headers: rootGetters.getHeaders})  
 .then(() =>  
 commit('setUsers',  
 [...state.users]  
 .filter(user => user.id !== user\_id )  
 ),  
 )  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 },  
 async changeRole({state, rootState, rootGetters}, obj){  
 let path = `${state.defaultRoot}/${obj.id}/change\_role`  
 let role = 'Admin'  
 if(obj.role === false)  
 role = 'User'  
 path += `?role\_name=${role}`  
 await instance  
 .patch(path)  
 .catch(error => {  
 console.log(error)  
 rootState.errors.push(error.response.data.detail)  
 })  
 },  
 async decodeRoleFromJWT({rootState}){

const payload = jwt\_decode(rootState.accessToken)  
 rootState.isAdmin = payload.role === 'Admin';  
 rootState.tokenExp = payload.exp  
 },  
 async getCurrentUser({state, commit, rootState, rootGetters}){  
 await commit('clearUser')  
 const path = `${state.defaultRoot}/me`  
 await instance  
 .get(path, {headers: rootGetters.getHeaders})  
 .then(response => {  
 commit('setUser', response.data)

})  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 },  
 async updateUser({state, rootState, rootGetters}){  
 const path = `${state.defaultRoot}/${state.user.id}/update`  
 rootState.errors = []  
 await instance  
 .put(path, state.user, {headers: rootGetters.getHeaders})  
 .catch(error => {  
 rootState.errors.push(error.response.data.detail)  
 })  
 },  
 },  
 namespaced: true  
  
}

*index.js :*

import {createStore} from 'vuex'  
import {*userModule*} from "./userModele"  
import {*bookModule*} from "./bookModule"  
import {*genreModule*} from "./genreModule"  
import {*basketModule*} from "./basketModule";  
import {*orderModule*} from "./orderModule";  
  
export default createStore({  
 state: {  
 isAuth: false,  
 isAdmin: false,  
 accessToken: '',  
 errors: [],  
 tokenExp: null,  
 },  
 mutations: {  
 setToken(*state*, *access*){  
 *state*.accessToken = *access* },  
 setAuth(*state*, *bool*) {  
 *state*.isAuth = *bool* },  
 setAdmin(*state*, *bool*){  
 *state*.isAdmin = *bool* },  
 setExp(*state*, *exp*){  
 *state*.tokenExp = *exp* },  
 clearErrors(*state*){  
 *state*.errors = []  
 }  
 },  
 getters: {  
 getHeaders(*state*){  
 return {  
 Authorization: `Bearer ${*state*.accessToken}`,  
 }  
 },  
  
 },  
 actions: {  
 async logout({*state*}){  
 *state*.accessToken = ''  
 *state*.isAuth = false  
 *state*.isAdmin = false  
 *state*.tokenExp = null  
 *state*.errors = []  
 *localStorage*.accessToken = *state*.accessToken  
 *localStorage*.isAuth = *state*.isAuth  
 *localStorage*.isAdmin = *state*.isAdmin  
 *localStorage*.tokenExp = *state*.tokenExp  
 *localStorage*.user\_id = null  
 },  
 },  
  
 modules: {  
 user: *userModule*,  
 book: *bookModule*,  
 genre: *genreModule*,  
 basket: *basketModule*,  
 order: *orderModule* }  
})

***Router :***

*instatnce.js :*

import *axios* from "axios";  
  
export const *instance* = *axios*.create({  
 baseURL: 'http://192.168.0.105:8000/api/',  
});

*router.js :*

import {createRouter, createWebHistory} from 'vue-router'  
import LoginPage from "@/pages/LoginPage";  
import RegisterPage from "@/pages/RegisterPage";  
import AdminPage from "@/pages/AdminPage";  
import AccountPage from "@/pages/AccountPage";  
import StoreBooksPage from "@/pages/StoreBooksPage";  
import StoreBookPage from "@/pages/StoreBookPage";  
import MyBooksPage from "@/pages/MyBooksPage";  
import OrdersPage from "@/pages/OrdersPage";  
import NotFoundPage from "@/pages/NotFoundPage";  
  
const routes = [  
 {  
 path: "/login",  
 component: LoginPage,  
 meta: { title: 'Log in' }  
 },  
 {  
 path: "/register",  
 component: RegisterPage,  
 meta: { title: 'Register' }  
 },  
 {  
 path: "/logout",  
 },  
 {  
 path: "/",  
 component: StoreBooksPage,  
 meta: { title: 'Shop' }  
 },  
 {  
 path: "/my/books",  
 component: MyBooksPage,  
 meta: { title: 'Books' }  
 },  
 {  
 path: "/books/:id",  
 component: StoreBookPage,  
 meta: { title: 'Book' }  
 },  
 {  
 path: "/orders",  
 component: OrdersPage,  
 meta: { title: 'Orders' }  
 },  
 {  
 path: "/admin",  
 component: AdminPage,  
 meta: { title: 'Admin' }  
 },  
 {  
 path: "/account",  
 component: AccountPage,  
 meta: { title: 'Account' }  
 },  
 {  
 path: "/:pathMatch(.\*)\*" ,  
 name: 'NotFound',  
 component: NotFoundPage,  
 meta: { title: '404 Not found' }  
 }  
]  
  
const *router* = createRouter({  
 routes,  
 history: createWebHistory(*process*.env.BASE\_URL)  
})  
  
*router*.beforeEach((*to*, *from*, *next*) => {  
 *document*.title = *to*.meta.title  
 *next*()  
});  
  
  
export default *router*

***Pages :***

*AccountPage.vue :*

<template>  
 <div class**="user\_\_page"**>  
 <user-form :modified**="**modified**"**>  
 <template v-slot:header>  
 <img src**="@/assets/user\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 **Account** </template>  
 <template v-slot:password>  
 **New password** </template>  
 <template v-slot:submit\_\_name>  
 **Save** </template>  
 </user-form>  
 </div>  
</template>  
  
<script>  
import {*mapActions*, *mapMutations*, *mapState*} from "vuex";  
import UserForm from "@/components/UserForm";  
  
export default {  
 name: "AccountPage",  
 components: {UserForm},  
 mounted() {  
 if(this.isAuth)  
 this.getCurrentUser()  
 else  
 this.$router.push('/login')  
 },  
 beforeUnmount() {  
 this.clearErrors()  
 },  
 data(){  
 return{  
 modified: true  
 }  
 },  
 computed: {  
 ...*mapState*({  
 isAuth: *state* => *state*.isAuth,  
 user: *state* => *state*.user.user,  
 }),  
 },  
 methods: {  
 ...*mapActions*({  
 getCurrentUser: 'user/getCurrentUser',  
 updateUser: 'user/updateUser',  
 }),  
 ...*mapMutations*({  
 clearErrors: 'clearErrors'  
 })  
 }  
}  
</script>  
  
<style scoped>  
  
</style>

*AdminPage.vue :*

<template>  
 <div class**="d-flex flex-column gap-5 m-auto"**>  
 <div>  
 <user-list>  
 </user-list>  
 </div>  
 <div class**="d-flex justify-content-center"**>  
 <genre-list>  
 </genre-list>  
 </div>  
 <div class**="btns d-flex justify-content-between"**>  
 <my-button @click**="**$router.back()**"**> **Back** </my-button>  
 <my-button @click**="**showGenreDialog**"**> **Add genre** </my-button>  
 </div>  
 <my-dialog v-model:show**="**genreDialogVisible**"**>  
 <create-genre-form>  
 <template v-slot:submit\_\_name>  
 **Add** </template>  
 </create-genre-form>  
 </my-dialog>  
 </div>  
</template>  
<script>  
import UserList from "../components/UserList";  
import {*mapActions*, *mapMutations*, *mapState*} from "vuex";  
import GenreList from "@/components/GenreList";  
import CreateGenreForm from "../components/CreateGenreForm";  
  
export default {  
 name: "AdminPage",  
 components: {CreateGenreForm, GenreList, UserList},  
 data(){  
 return{  
 genreDialogVisible: false,  
 }  
 },  
 beforeRouteEnter(*to*, *from*, *next*){  
 *next*(*vm* => {  
 if(!(*vm*.isAdmin && *vm*.isAuth)){  
 *vm*.$store.state.errors.push('You are not an admin')  
 *vm*.$router.push('/login')  
 }  
 })  
 },  
 beforeUnmount() {  
 this.clearUsers()  
 this.clearGenres()  
 },  
 methods: {  
 ...*mapMutations*({  
 clearErrors: 'clearErrors',  
 clearGenres:'genre/clearGenreStore',  
 clearUsers: 'user/clearUsers',  
 }),  
 showGenreDialog(){  
 this.genreDialogVisible = true  
 },  
  
 },  
 computed: {  
 ...*mapState*({  
 isAdmin: *state* => *state*.isAdmin,  
 isAuth: *state* => *state*.isAuth,  
 users: *state* => *state*.user.users,  
 genres: *state* => *state*.genre.genres,  
 genre: *state* => *state*.genre.genre,  
 }),  
 },  
}  
</script>  
  
<style scoped>  
</style>

*LoginPage.vue :*

<template>  
 <div class**="user\_\_page"**>  
 <Form v-slot**="**{ handleSubmit }**"** :validation-schema**="**schema**"** as**="div"** class**="user\_\_form"**>  
 <my-title class**="title"**>  
 <img src**="@/assets/login\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 **Log in** </my-title>  
 <my-error-list :errors**="**errors**"**></my-error-list>  
 <form @submit**="**handleSubmit($event, login)**"** class**="form"**>  
 <MyField  
 v-model**="**user.email**"** name**="email"** placeholder**="email@gmail.com"** />  
 <MyErrorMessage name**="email"** />  
 <MyField  
 v-model**="**user.password**"** name**="password"** type**="password"** placeholder**="password"** />  
 <MyErrorMessage name**="password"** />  
 <my-button  
 type**="submit"** >  
 **Log in** </my-button>  
 </form>  
 <p>  
 **I don't have an account :** <router-link to**="/register"** class**="link"**>**Sign up**</router-link>  
 </p>  
 </Form>  
 </div>  
</template>  
  
<script>  
import {mapActions, mapMutations, mapState} from "vuex";  
import {Form} from 'vee-validate'  
  
import \* as yup from 'yup'  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import MyErrorList from "@/components/UI/MyErrorList";  
import MyTitle from "../components/UI/MyTitle";  
  
export default {  
 name: "LoginPage",  
 components:{  
 MyTitle,  
 MyErrorList,  
 MyField,  
 Form, MyErrorMessage  
 },  
 async beforeUnmount() {  
 if(this.isAuth) {  
 await this.getCurrentUser()  
 localStorage.accessToken = this.accessToken  
 localStorage.refreshToken = this.refreshToken  
 localStorage.isAuth = this.isAuth  
 localStorage.isAdmin = this.isAdmin  
 localStorage.tokenExp = this.tokenExp  
 localStorage.user\_id = this.user.id  
 }  
 this.clearErrors()  
 },  
 computed: {  
 ...mapState({  
 accessToken: state => state.accessToken,  
 refreshToken: state => state.refreshToken,  
 isAuth: state => state.isAuth,  
 isAdmin: state => state.isAdmin,  
 user: state => state.user.user,  
 tokenExp: state => state.tokenExp,  
 errors: state => state.errors  
 }),  
 schema() {  
 return yup.object().shape({  
 email: yup.string().email().max(50).required().label('Email'),  
 password: yup.string().min(5).required().label('Password'),  
 })  
 }  
 },  
 methods: {  
 ...mapActions({  
 login: 'user/login',  
 getCurrentUser: 'user/getCurrentUser'  
 }),  
 ...mapMutations({  
 clearErrors: 'clearErrors'  
 })  
  
 }  
}  
  
</script>  
  
<style scoped>  
  
</style>

*MyBooksPage.vue :*

<template>  
 <div>  
 <custom-books-page :owner**="**true**"**>  
 <template v-slot:title>  
 <img src**="@/assets/book\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 **Your Books** </template>  
 <template v-slot:create>  
 <my-button @click**="**showDialog**"**> **Add book** </my-button>  
 <my-dialog v-model:show**="**dialogVisible**"**>  
 <book-form>  
 <template v-slot:submit\_\_name>  
 **Add** </template>  
 </book-form>  
 </my-dialog>  
 </template>  
 </custom-books-page>  
 </div>  
</template>  
  
<script>  
import CustomBooksPage from "@/components/CustomBooksPage";  
import BookForm from "@/components/BookForm";  
import {*mapState*} from "vuex";  
  
export default {  
 name: "StoreBooksPage",  
 components: {  
 BookForm,  
 CustomBooksPage  
 },  
 data(){  
 return{  
 dialogVisible: false  
 }  
 },  
 beforeRouteEnter(*to*, *from*, *next*){  
 *next*(*vm* => {  
 if(!*vm*.isAuth) {  
 *vm*.errors.push('You are not authorized')  
 *vm*.$router.push('/login')  
 }  
 })  
 },  
 methods: {  
 async showDialog() {  
 this.dialogVisible = true  
 },  
 },  
 computed: {  
 ...*mapState*({  
 isAuth: *state* => *state*.isAuth,  
 isAdmin: *state* => *state*.isAdmin,  
 errors: *state* => *state*.errors  
 })  
 }  
}  
</script>  
  
<style scoped>  
</style>

*NotFoundPage.vue :*

<template>  
 <div class**="page"**>  
 <img src**="@/assets/error\_404\_negate.png"** height**="100"** width**="100"**/>  
 <span>**Page not found**</span>  
 </div>  
</template>  
  
<script>  
export default {  
 name: "NotFoundPage"  
}  
</script>  
  
<style scoped>  
.page{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *align-items*: **center**;  
 *justify-content*: **center**;  
 *height*: 90**vh**;  
 *gap*: 3**px**;  
}  
</style>

*OrdersPage.vue :*

<template>  
 <div>  
 <order-list :is-admin**="**isAdmin**"**>  
 <template v-slot:image>  
 <img src**="@/assets/sent\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 </template>  
 </order-list>  
 </div>  
</template>  
  
<script>  
import OrderList from "@/components/OrderList";  
import {*mapActions*, *mapMutations*, *mapState*} from "vuex";  
export default {  
 name: "OrdersPage",  
 components: {OrderList},  
 beforeRouteEnter(*to*, *from*, *next*){  
 *next*(*vm* => {  
 if(!*vm*.isAuth) {  
 *vm*.errors.push('You are not authorized')  
 *vm*.$router.push('/login')  
 }  
 })  
 },  
 beforeUnmount() {  
 this.clearOrders()  
 },  
 computed: {  
 ...*mapState*({  
 isAuth: *state* => *state*.isAuth,  
 isAdmin: *state* => *state*.isAdmin,  
 orders: *state* => *state*.order.orders,  
 errors: *state* => *state*.errors  
 })  
 },  
 methods: {  
 ...*mapActions*({  
 getOrders: 'order/getOrders'  
 }),  
 ...*mapMutations*({  
 clearOrders: 'order/clearOrderStore'  
 })  
 }  
}  
</script>  
  
<style scoped>  
  
</style>

*RegisterPage.vue :*

<template>  
 <div class**="user\_\_page"**>  
 <user-form>  
 <template v-slot:header>  
 <img src**="@/assets/personal\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 **Register** </template>  
 <template v-slot:password>  
 **Password** </template>  
 <template v-slot:submit\_\_name>  
 **Register** </template>  
 <template v-slot:link>  
 <p>  
 **I already have an account :** <router-link to**="/login"** class**="link"**>**Log in**</router-link>  
 </p>  
 </template>  
 </user-form>  
 </div>  
</template>  
  
<script>  
import {mapMutations} from "vuex";  
import UserForm from "@/components/UserForm";  
  
export default {  
 name: "RegisterPage",  
 components: {UserForm},  
 mounted() {  
 this.clearUser()  
 },  
 beforeUnmount() {  
 this.clearErrors()  
 },  
 methods: {  
 ...mapMutations({  
 clearUser: 'user/clearUser',  
 clearErrors: 'clearErrors'  
 }),  
 },  
}  
</script>  
  
<style scoped>  
  
</style>

*StoreBookPage.vue :*

<template>  
 <div class**="item"**>  
 <div class**="d-flex justify-content-center w-100 top-content"**>  
 <img class**="image align"** v-image-observer:[book.image]**="**getImage**"** :key**="**book.id**"** alt**="Card image cap"** @click**="**showImageDialog**"**>  
 <div class**="p-4 short\_\_info"**>  
 <div class**="mb-2"**>  
 <h5 class**="text-center mb-4 title"**>**{{book.name}}**</h5>  
 <p>**Author : {{book.author}}**</p>  
 <p>**Price : ${{book.price}}**</p>  
 <div>  
 <p v-if**="**book.count > 0**"**>**Available : {{book.count}} pcs.**</p>  
 <p v-else class**="error"**>**Not available**</p>  
 </div>  
 </div>  
 </div>  
 </div>  
 <div>  
 <div class**="d-flex justify-content-between pe-lg-5 ps-lg-5 mt-3"**>  
 <div class**="d-flex flex-column justify-content-evenly"**>  
 <span>**Publication : {{ new Date(book.publication\_date).toLocaleDateString()}}**</span>  
 <span>**Last update date : {{new Date(book.update\_date).toLocaleDateString()}}**</span>  
 <span id**="ISBN"**>**ISBN : #{{book.ISBN}}**</span>  
 </div>  
 <div class**="d-flex content"**>  
 <div v-if**="**book.count > 0 && isAuth**"** class**="d-flex justify-content-center"**>  
 <form @submit.prevent**="**action**"** method**="post"** class**="order\_\_form addToBasket gap-2 p-2"**>  
 <my-input v-model**="**order.count**"** type**="number"** id**="count"**/>  
 <div v-if**="**isCartLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button v-else type**="submit"**>**Add to card**</my-button>  
 </form>  
 </div>  
 </div>  
 </div>  
 <div v-if**="**book.genres.length > 0**"** class**="m-auto"**>  
 <p class**="text-center mb-3"**>  
 **Genres** </p>  
 <p class**="row justify-content-center"**>  
 <span v-for**="**genre in book.genres**"** :key**="**genre.name**"** class**="text-center genre m-1 col-auto"**>**{{genre.name}}**</span>  
 </p>  
 </div>  
 <div v-if**="**book.content.length > 0**"** class**="mt-3"**>  
 <p class**="align-self-center text-center"**>**Plot**</p>  
 <hr/>  
 <p class**="plot"**>**{{book.content}}**</p>  
 <hr/>  
 </div>  
 </div>  
 <div class**="d-flex justify-content-between mt-3 gap-3"**>  
 <my-button @click**="**$router.back()**"**>**Back**</my-button>  
 <div class**="d-flex gap-3"** v-if**="**isCreator**"**>  
 <my-button @click**="**showChangeGenresDialog**"**>**Change genres**</my-button>  
 <my-button @click**="**showBookUpdateDialog**"**>**Update**</my-button>  
 <my-button @click**="**showImageDialog**"**>**Change image**</my-button>  
 </div>  
 </div>  
 <my-dialog v-model:show**="**imageDialogVisible**"**>  
 <Form v-slot**="**{ *handleSubmit* }**"** as**="div"**>  
 <my-error-list :errors**="**errors**"** class**="text-center text-break"**></my-error-list>  
 <form @submit**="***handleSubmit*(*$event*, changeBookImage)**"** enctype**="multipart/form-data"** method**="post"** id**="uploadForm"** class**="form"**>  
 <label for**="files"** class**="label text-break mb-2"** id**="file\_label"**>**Select image**</label>  
 <my-field type**="file"** name**="file"** id**="files"** v-focus class**="file"** @change**="**changeFileStatus**"** accept**="image/\*"** required/>  
 <my-error-message name**="file"**/>  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button v-else type**="submit"**>**Submit**</my-button>  
 </form>  
 </Form>  
 </my-dialog>  
 <my-dialog v-model:show**="**updateBookDialogVisible**"**>  
 <book-form :modified**="**true**"**>  
 <template v-slot:submit\_\_name>  
 **Save** </template>  
 </book-form>  
 </my-dialog>  
 <my-dialog v-model:show**="**changeGenresDialogVisible**"**>  
 <genre-form></genre-form>  
 </my-dialog>  
 </div>  
</template>  
  
<script>  
import BookItem from "@/components/BookItem";  
import {mapActions, mapMutations, mapState} from "vuex";  
import BookForm from "@/components/BookForm";  
import MyErrorList from "@/components/UI/MyErrorList";  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import {Form} from "vee-validate";  
import GenreForm from "@/components/GenreForm";  
  
  
export default {  
 name: "StoreBookPage",  
 components: {GenreForm, BookForm, BookItem, MyErrorList, MyField, MyErrorMessage, Form},  
 async beforeMount() {  
 await this.getBook(this.$router.currentRoute.value.params.id)  
 if(this.book.count > 0 && this.isAuth) {  
 this.setBookLimits()  
 }  
 this.isCreator = this.book.owner\_id === JSON.parse(localStorage.user\_id) || this.isAdmin  
 },  
 data(){  
 return{  
 updateBookDialogVisible: false,  
 imageDialogVisible: false,  
 changeGenresDialogVisible: false,

isCreator: false  
 }  
 },  
 computed: {  
 ...mapState({  
 isAuth: state => state.isAuth,  
 isAdmin: state => state.isAdmin,  
 book: state => state.book.book,  
 order: state => state.basket.order,  
 errors: state => state.errors,  
 isLoading: state => state.book.isLoading,  
 isCartLoading: state => state.basket.isLoading  
 }),  
 },  
 async beforeUnmount() {  
 await this.clearBook()  
 },  
 methods: {  
 ...mapActions({  
 getBook: 'book/getBook',  
 getImage: 'book/getBookImage',  
 addToBasket: 'basket/addToBasket',  
 changeImage: 'book/changeBookImage'  
 }),  
 ...mapMutations({  
 clearBook: 'book/clearBook'  
 }),  
 async action(){  
 if(!this.isAuth)  
 this.$router.push('/login')  
 else {  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The book has been successfully added to your cart</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1500  
 })  
 await this.addToBasket(this.book)  
 if(this.book.count > 0)  
 this.setBookLimits()  
 }  
 },  
 async showBookUpdateDialog(){  
 this.updateBookDialogVisible = true  
 },  
 async showImageDialog(){  
 if(this.isCreator)  
 this.imageDialogVisible = true  
 },  
 async showChangeGenresDialog(){  
 this.changeGenresDialogVisible = true  
 },  
 setBookLimits(){  
 const order\_count = document.querySelector('#count')  
 order\_count.setAttribute('min', 1)  
 order\_count.setAttribute('max', this.book.count)  
 },  
 changeFileStatus(event){  
 const image = event.target  
 const label = document.querySelector('#file\_label')  
 if(image.files.length === 0)  
 label.innerHTML = `Select image`

else  
 label.innerHTML = `${image.files[0].name}`  
 },  
 async changeBookImage(){  
 const dialog = document.querySelector('.dialog')  
 const image = document.querySelector('.image')  
 await this.changeImage()  
 await this.getImage({target: image, image\_name: this.book.image})  
 if(this.errors.length === 0){  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The image has been successfully changed</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 }  
 }  
 },  
  
}  
</script>  
  
<style scoped>  
.item{  
 *background-color*: **inherit**;  
 *border*:**none**;  
 *width*: 60**vw**;  
 *min-height*: 90**vh**;  
 *margin*: **auto**;  
 *overflow-wrap*: **break-word**;  
}  
.image{  
 *width*: 300**px**;  
 *max-height*: 450**px**;  
 *border-radius*: 15**px**;  
}  
.top-content{  
 *display*: **grid**;  
 *grid-template-columns*: 25**vw** 10**vw**;  
}  
p{  
 *margin*: 0;  
 *line-height*: 27**px**;  
}  
  
.order\_\_form>button, .order\_\_form>input {  
 *margin*: 0;  
 *width*: 100%;  
}  
  
.order\_\_form{  
 *width*: 15**vw**;  
 *max-width*: 200**px**;  
 *min-width*: 150**px**;  
 *display*: **flex**;  
 *align-items*: **center**;  
 *flex-direction*: **column**;  
}  
  
.short\_\_info{  
 *display*: **flex**;

*max-width*: 25**vw**;  
 *flex-direction*: **column**;  
 *justify-content*: **space-around**;  
}  
  
.file{  
 *color*: #FFF587;  
 *width*: 0.1**px**;  
 *height*: 0.1**px**;  
 *opacity*: 0;  
 *overflow*: **hidden**;  
 *position*: **absolute**;  
 *z-index*: -1;  
}  
  
.label {  
 *width*: 180**px**;  
 *border-radius*: 4**px**;  
 *text-align*: **center**;  
 *cursor*: **pointer**;  
 *font*: 14**px**/50**px** 'Merriweather', **serif**;  
 *transition*: **all** 0.3**s ease-in-out**;  
 *color*: #FFF587;  
 *line-height*: 22**px !important**;  
  
}  
  
.label:hover {  
 *color*: #ffffff;  
}  
  
.genre{  
 *background-color*: rgba(149, 149, 149, 0.34);  
}  
  
.title{  
 *font-weight*: **bold**;  
 *font-style*: **italic**;  
}  
  
.plot{  
 *text-align*: **justify**;  
}  
  
.addToBasket{  
 *background-color*: rgba(112, 114, 247, 0.2);  
 *border-radius*: 5**px**;  
 *border*: **solid** 2**px** rgba(112, 114, 247, 0.4);  
 *height*: **fit-content**;  
}  
  
@media **screen** and (*max-width*: 900**px**){  
 .item>div{  
 *flex-direction*: **column**;  
 *align-items*: **center**;  
 }  
 .image{  
 *justify-self*: **center**;  
 }  
 .short\_\_info{  
 *max-width*: 100**vw**;  
 *width*: 100%;  
 *padding*: 0 **!important**;  
 }

.item>div:last-child{  
 *gap*: 0.5**rem**;  
 *flex-direction*: **column-reverse**;  
 *align-items*: **center**;  
 }  
 .item>div:last-child>div{  
 *gap*: 0.5**rem !important**;  
 *width*: 100%;  
 *flex-direction*: **column**;  
 }  
 .item>div:last-child button{  
 *width*: 100%;  
 }  
}  
  
@media **screen** and (*max-width*: 750**px**){  
 .item{  
 *margin*: 0 **!important**;  
 *width*: 100%;  
 }  
}  
  
@media **screen** and (*max-width*: 650**px**) {  
  
 .item>div:nth-child(2)>div:first-child{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *margin*: 0 **!important**;  
 *gap*: 0.6**rem**;  
 }  
  
 .order\_\_form{  
 *width*: 60**vw**;  
 *max-width*: 60**vw**;  
 }  
  
 .content{  
 *justify-content*: **center**;  
 }  
  
  
}  
img:hover{  
 *cursor*: **pointer**;  
}  
</style>

*StoreBooksPage.vue :*

<template>  
 <div>  
 <custom-books-page>  
 <template v-slot:title>  
 <img src**="@/assets/shop\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 **Shop** </template>  
 </custom-books-page>  
 </div>  
</template>  
  
<script>  
import CustomBooksPage from "@/components/CustomBooksPage";  
import {*mapState*} from "vuex";  
  
export default {  
 name: "StoreBooksPage",  
 components: {  
 CustomBooksPage  
 },  
 computed: {  
 ...*mapState*({  
 isAuth: *state* => *state*.isAuth,  
 isAdmin: *state* => *state*.isAdmin,  
 })  
 }  
}  
</script>  
  
<style scoped>  
</style>

*toggleDialogMixin.js :*

export default {  
 props: {  
 show:{  
 type: *Boolean*,  
 default: false  
 }  
 },  
 methods: {  
 hideDialog(){  
 this.$emit('update:show', false)  
 }  
 },  
  
}

***Directives :***

*directives.js :*

import VIntersection from "./VIntersection";  
import VFocus from "./VFocus";  
import VImageObserver from "./VImageObserver";  
  
  
  
export default [  
 VIntersection,  
 VFocus,  
 VImageObserver  
]

*VFocus.js :*

export default {  
 name: 'focus',  
 mounted(el){  
 el.focus()  
 }  
}

*VImageObserver.js :*

export default {  
 name: 'image-observer',  
 mounted(el, binding){  
  
 const options = {  
 rootMargin: '0px',  
 threshold: 1.0  
 }  
 const name = binding.arg  
 const callback = (entries) =>{  
 if(el.src === ""){  
 binding.value({target: el, image\_name: name})  
 }  
 }  
  
 const observer = new IntersectionObserver(callback, options)  
 observer.observe(el)  
 },  
}

*VIntersection.js :*

export default {  
 name: 'intersection',  
 mounted(*el*, *binding*){  
  
 const options = {  
 rootMargin: '0px',  
 threshold: 1.0  
 }  
  
 const owner = *binding*.arg  
 const callback = (*entries*) =>{  
 if(*entries*[0].isIntersecting){  
 *binding*.value(owner)  
 }  
 }  
  
 const observer = new *IntersectionObserver*(callback, options)  
 observer.observe(*el*)  
 },  
}

***Components :***

*BasketBook.vue :*

<template>  
 <div :key**="**order.id**"** class**="d-flex justify-content-between item"**>  
 <div>  
 <img class**="image"** v-image-observer:[order.image]**="**getImage**"** alt**="Card image cap"**>  
 </div>  
 <div class**="short\_\_info p-2"**>  
 <h5 class**="title"**>**{{order.name}}**</h5>  
 <p>**{{order.author}}**</p>  
 <p>**${{order.price}}**</p>  
 <p>**Count : {{order.order\_count}}**</p>  
 </div>  
 <div class**="d-flex align-items-center"**>  
 <div class**="d-flex flex-column text-center price p-2 gap-2"**>

<span>**${{ Number(order.price \* order.order\_count).toFixed(2)}}**</span>  
 <my-button class**="h-25"** @click.once**="**removeBook({id: order.id, count: order.order\_count})**"**>**Remove**</my-button>  
 </div>  
 </div>  
 </div>  
</template>  
  
<script>  
import {*mapActions*, *mapState*} from "vuex";  
import MyButton from "./UI/MyButton";  
  
export default {  
 name: "BasketBook",  
 components: {MyButton},  
 props: {  
 order: {  
 type: *Object*,  
 required: true  
 }  
 },  
 data(){  
 return {  
 basketBook: this.book  
 }  
 },  
 computed: {  
 ...*mapState*({  
 book: *state* => *state*.book.book  
 })  
 },  
 methods: {  
 ...*mapActions*({  
 getBook: 'book/getBook',  
 getImage: 'book/getBookImage',  
 removeBook: 'basket/removeFromBasket'  
 }),  
 }  
}  
</script>  
  
<style scoped>  
.image{  
 *width*: 200**px**;  
 *height*: 200**px**;  
 *border-radius*: 10**px**;  
}  
  
.short\_\_info{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *justify-content*: **space-around**;  
 *margin-right*: **auto**;  
}  
.title{  
 *text-align*: **justify**;  
 *word-break*: **break-word**;  
}  
  
div.price{  
 *background-color*: rgba(112, 114, 247, 0.2);  
 *border-radius*: 5**px**;  
 *border*: **solid** 2**px** rgba(112, 114, 247, 0.4);  
}

@media **screen** and (*max-width*: 1400**px**){  
 .item{  
 *flex-direction*: **column**;  
 *font-size*: 1**rem**;  
 }  
 .item>div:first-child{  
 *display*: **flex**;  
 *justify-content*: **center**;  
 }  
 .price{  
 *width*: 100%;  
 }  
  
}  
</style>

*BookForm.vue :*

<template>  
 <Form v-slot**="**{ handleSubmit }**"** :validation-schema**="**schema**"** as**="div"** class**="dialog"**>  
 <my-error-list :errors**="**errors**"**></my-error-list>  
 <form @submit**="**handleSubmit($event, action)**"** method**="post"** class**="d-flex flex-column"** name**="book\_form"**>  
 <div class**="scroll d-flex flex-column justify-content-between"** >  
 **Title :** <my-field  
 v-model**="**book.name**"** name**="name"** v-focus  
 />  
 <my-error-message name**="name"** />  
 **Author :** <my-field  
 v-model**="**book.author**"** name**="author"** />  
 <my-error-message name**="author"** />  
 **Count :** <my-field  
 v-model**="**book.count**"** name**="count"** />  
 <my-error-message name**="count"** />  
 **Price :** <my-field  
 v-model**="**book.price**"** name**="price"** />  
 <my-error-message name**="price"** />  
 **ISBN :** <my-field  
 v-model**="**book.ISBN**"** name**="ISBN"** />  
 <my-error-message name**="ISBN"** />  
 **Date of publication :** <my-field  
 v-model**="**book.publication\_date**"** type**="date"** class**="w-100"** name**="publication"** />  
 <my-error-message name**="publication"** />  
 **Description :** <textarea  
 v-model**="**book.content**"** name**="content"** maxlength**="5000"** ></textarea>  
 <my-error-message name**="content"** />  
 </div>  
 <div class**="mt-2 d-flex justify-content-center"**>  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button  
 v-else  
 type**="submit"** class**="w-100"** >  
 <slot name**="submit\_\_name"**></slot>  
 </my-button>  
 </div>  
 </form>  
 </Form>  
</template>  
  
<script>  
import {mapActions, mapMutations, mapState} from "vuex";  
import {Form} from 'vee-validate'  
  
import \* as yup from 'yup'  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import MyErrorList from "./UI/MyErrorList";  
import router from "../router/router";  
  
  
export default {  
 name: "BookForm",  
 components: {Form, MyField, MyErrorList, MyErrorMessage},  
 props: {  
 modified:{  
 type: Boolean,  
 default: false  
 },  
 },  
 mounted() {  
 this.clearErrors()  
 },  
 methods: {  
 ...mapMutations({  
 clearErrors: 'clearErrors'  
 }),  
 ...mapActions({  
 createBook: 'book/createBook',  
 updateBook: 'book/updateBook'  
 }),  
 async action(){  
 const dialog = document.querySelector('.dialog')  
 if(this.modified) {  
 await this.updateBook(this.book.id)  
 if(this.errors.length === 0)  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The book has been successfully updated</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1500

})  
 dialog.click()  
  
 }  
 else {  
 await this.createBook()  
 if(this.errors.length === 0)  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The book has been successfully created</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1500  
 })  
 dialog.click()  
 }  
 },  
 },  
 computed: {  
 ...mapState({  
 book: state => state.book.book,  
 errors: state => state.errors,  
 isLoading: state => state.book.isFormLoading,  
 }),  
 schema() {  
 return yup.object().shape({  
 name: yup.string().max(100).required().label('Title'),  
 author: yup.string().max(50).required().label('Author'),  
 count: yup.number().typeError("Count is a number field").integer('Invalid decimal').min(1).required().label('Count'),  
 price: yup.number().typeError("Count is a number field").min(1).required().label('Price'),  
 ISBN: yup.string().matches('^(?=(?:\\D\*\\d){10}(?:(?:\\D\*\\d){3})?$)[\\d-]+$', 'Enter the correct ISBN').required().label('ISBN'),  
 publication: yup.date().min(new Date(1)).required().label('Date of publication'),  
 content: yup.string().max(5000).label('Description'),  
 })  
 },  
 },  
}  
</script>  
  
<style scoped>  
.dialog{  
 *width*: 50**vw**;  
 *max-width*: 500**px**;  
 *min-width*: 350**px**;  
}  
  
.scroll{  
 *height*: 85**vh**;  
 *min-height*: 300**px**;  
 *overflow-y*: **auto**;  
 *overflow-x*: **hidden**;  
}  
  
textarea{  
 *color*: #292a5c;  
 *height*: 20**vh**;

*min-height*: 100**px**;  
}  
  
</style>

*BookItem.vue :*

<template>  
 <div class**="card item bg-gradient"** :class**="**{ 'gray' : book.count === 0}**"** @click**="**$router.push(`/books/${book.id}`)**"**>  
 <div>  
 <img class**="image"** v-image-observer:[book.image]**="**getImage**"** alt**="Book image"**>  
 </div>  
 <div class**="card-body item-body"**>  
 <h5 class**="card-title"**>**{{book.name}}**</h5>  
 <p class**="card-text text"**>  
 <span>**{{book.author}}**</span>  
 <span>**${{book.price}}**</span>  
 <span id**="ISBN"**>**#{{book.ISBN}}**</span>  
 </p>  
 <my-button @click.stop**="**$emit('remove', book.id)**"** v-if**="**isCreator**"** class**="w-50 align-self-center"**>  
 **Delete** </my-button>  
 </div>  
 </div>  
</template>  
  
<script>  
import {*mapActions*} from "vuex";  
  
export default {  
 name: "BookItem",  
 props:{  
 book: {  
 type: *Object*,  
 required: true,  
 },  
 isCreator: {  
 type: *Boolean*,  
 default: false  
 }  
 },  
 methods: {  
 ...*mapActions*({  
 getImage: 'book/getBookImage'  
 }),  
 }  
  
}  
</script>  
  
<style scoped>  
.item{  
 *width*: 17**rem**;  
 *background-color*: rgba(149, 149, 149, 0.3);  
 *border*: **none**;  
}  
.item:hover{  
 *cursor*: **pointer**;  
}  
.item-body{  
 *height*: 100%;  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *justify-content*: **space-between**;  
}  
#ISBN{  
 *font-size*: 12**px**;  
 *color*: #909090;  
}  
.image{  
 *width*: 100%;  
 *max-height*: 380**px**;  
 *min-height*: 380**px**;  
 *border-radius*: 3**px** 3**px** 0 0;  
 *filter*: saturate(107%);  
}  
  
.image:hover{  
 *cursor*: **pointer**;  
}  
.text{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
}  
  
.gray{  
 *filter*: opacity(.8) brightness(.5) saturate(.2);  
}  
  
</style>

*BookList.vue :*

<template>  
 <div v-if**="**books.length > 0**"**>  
 <div class**="books"**>  
 <transition-group name**="list"**>  
 <book-item  
 v-for**="**book in books**"** :key**="**book.id**"** :book**="**book**"** :is-creator**="**isCreator**"** class**="mb-3 m-2 text-center"** @remove**="**$emit('remove', book.id)**"** />  
 </transition-group>  
 </div>  
 </div>  
 <empty-list v-else>**No books**</empty-list>  
</template>  
  
<script>  
import BookItem from "./BookItem";  
  
export default {  
 name: "BookList",  
 components: {BookItem},  
 props: {  
 books: {  
 type: Array,  
 required: true  
 },  
 isCreator: {  
 type: Boolean,  
 default: false  
 }  
 },  
}  
</script>  
  
<style scoped>  
.books{  
 *padding*: 15**px**;  
 *display*: **flex**;  
 *flex-flow*: **row wrap**;  
 *justify-content*: **center**;  
 *align-items*: **stretch**;  
}  
  
@media **screen** and (*max-width*: 500**px**) {  
 .books{  
 *padding*: 15**px** 0 **!important**;  
 }  
}  
</style>

*CreateGenreForm.vue :*

<template>  
 <Form v-slot**="**{ handleSubmit }**"** :validation-schema**="**schema**"** as**="div"** class**="genre\_\_form"**>  
 <my-error-list :errors**="**errors**"**></my-error-list>  
 <form @submit**="**handleSubmit($event, action)**"** method**="post"** class**="form"**>  
 **Genre name :** <my-field  
 v-model**="**genre.name**"** name**="title"** />  
 <my-error-message name**="title"** />  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button  
 v-else  
 type**="submit"** @click.once  
 >  
 <slot name**="submit\_\_name"**></slot>  
 </my-button>  
 </form>  
 </Form>  
</template>  
  
<script>  
import {mapActions, mapMutations, mapState} from "vuex";  
import {Form} from 'vee-validate'  
  
import \* as yup from 'yup'  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import MyErrorList from "./UI/MyErrorList";  
import MyTitle from "./UI/MyTitle";  
  
export default {  
 name: "CreateGenreForm",  
 components: {  
 MyTitle,  
 MyErrorList,  
 Form, MyField, MyErrorMessage  
 },  
 props: {  
 modified:{  
 type: Boolean,  
 default: false  
 }  
 },  
 beforeUnmount() {  
 this.clearGenre()  
 this.clearErrors()  
 },  
 methods: {  
 ...mapActions({  
 createGenre: 'genre/createGenre',  
 updateGenre: 'genre/updateGenre',  
 }),  
 ...mapMutations({  
 clearGenre: 'genre/clearGenre',  
 clearErrors: 'clearErrors'  
 }),  
 async action(){  
 const dialog = document.querySelector('.dialog')  
 if(this.modified) {  
 await this.updateGenre()  
 if(this.errors.length === 0) {  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The genre has been successfully updated</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 }  
 }  
 else {  
 await this.createGenre()  
 if(this.errors.length === 0) {  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>The genre has been successfully created</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 }  
 }  
 }  
  
 },  
 computed: {  
 ...mapState({  
 genre: state => state.genre.genre,  
 errors: state => state.errors,  
 isLoading: state => state.genre.isLoading  
 }),  
 schema() {  
 return yup.object().shape({  
 title: yup.string().max(30).required().label('Genre name'),  
 })

},  
 },  
}  
</script>  
  
<style scoped>  
.genre\_\_form{  
 *width*: 25**vw**;  
 *min-width*: 200**px**;  
}  
  
</style>

*CustomBooksPage.vue :*

<template>  
 <div>  
 <my-title><slot name**="title"**></slot></my-title>  
 <div class**="app\_\_btns d-flex flex-wrap"**>  
 <div class**="sort"**>  
 <my-select  
 :model-value**="**selectedSort**"** @update:model-value**="**setSelectedSort**"** :options**="**sortOptions**"** @change**="**getBookList(owner)**"** />  
 <my-select  
 :model-value**="**selectedGenre**"** @update:model-value**="**setSelectedGenresSort**"** :options**="**genresOptions**"** @change**="**getBookList(owner)**"** />  
 <div class**="d-flex align-content-center justify-content-center"**>  
 <input type**="checkbox"** id**="switch"** v-model**="**reverseSort.value**"** :value**="**reverseSort.value**"** @change**="**reverseBooks**"** />  
 <label for**="switch"**></label>  
 </div>  
 </div>  
 <slot name**="create"**></slot>  
 </div>  
 <book-list  
 :books**="**sortedBooks**"** v-if**="**!isLoading**"** :is-creator**="**owner**"** @remove**="**removeBook**"** />  
 <div v-else class**="spinner-grow"**>  
 </div>  
 <div  
 v-intersection:[owner]**="**getBookList**"** class**="observer"** >  
 </div>  
 </div>  
</template>  
  
<script>  
import {*mapActions*, *mapGetters*, *mapMutations*, *mapState*} from "vuex";  
import BookList from "./BookList";  
export default {  
 name: "BooksPage",  
 components: {  
 BookList,  
 },  
 props: {  
 owner: {  
 type: *Boolean*,  
 default: false  
 }  
 },  
 beforeMount() {  
 this.getAllGenres()  
 },  
 beforeUnmount() {  
 this.clearBookStore()  
 },  
 methods: {  
 ...*mapMutations*({  
 setSearchQuery: 'book/setSearchQuery',  
 setSelectedSort: 'book/setSelectedSort',  
 setSelectedGenresSort: 'genre/setSelectedSort',  
 clearBookStore: 'book/clearBookStore',  
 clearErrors: 'clearErrors',  
 setReverseSort: 'book/setReverseSort',  
 clearBooks: 'book/clearBooks',  
 }),  
 ...*mapActions*({  
 getBookList: 'book/getSortedBookList',  
 getAllGenres: 'genre/getAllGenres',  
 removeBook: 'book/removeBook'  
 }),  
 reverseBooks(){  
 this.clearBooks()  
 this.getBookList()  
 }  
 },  
 computed: {  
 ...*mapState*({  
 isAuth: *state* => *state*.isAuth,  
 books: *state* => *state*.book.books,  
 isLoading: *state* => *state*.book.isLoading,  
 selectedSort: *state* => *state*.book.selectedSort,  
 selectedGenre: *state* => *state*.genre.selectedSort,  
 searchQuery: *state* => *state*.book.searchQuery,  
 sortOptions: *state* => *state*.book.sortOptions,  
 genresOptions: *state* => *state*.genre.sortOptions,  
 reverseSort: *state* => *state*.book.reverseSort  
 }),  
 ...*mapGetters*({  
 sortedBooks: 'book/sortedBooks',  
 }),  
 },  
}  
</script>  
  
<style scoped>  
  
input[type=checkbox]{  
 *height*: 0;  
 *width*: 0;  
 *visibility*: **hidden**;  
}  
  
label {  
 *cursor*: **pointer**;

*width*: 40**px**;  
 *height*: 20**px**;  
 *background*: rgba(149, 149, 149, 0.34);  
 *display*: **block**;  
 *border-radius*: 100**px**;  
 *position*: **relative**;  
}  
label:after {  
 *content*: '';  
 *position*: **absolute**;  
 *top*: 0;  
 *left*: 0;  
 *width*: 20**px**;  
 *height*: 20**px**;  
 *background*: #fff;  
 *border-radius*: 90**px**;  
 *transition*: 0.3**s**;  
}  
  
input:checked + label {  
 *background*: #7072F7;  
}  
  
input:checked + label:after {  
 *left*: 100%;  
 *transform*: translateX(-100%);  
}  
  
label:active:after {  
 *width*: 70%;  
}  
  
.sort{  
 *display*: **flex**;  
 *gap*: 0.5**rem**;  
 *align-items*: **center**;  
}  
  
@media **screen** and (*max-width*: 450**px**){  
 .sort{  
 *width*: 100%;  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 }  
 .sort>\*{  
 *width*: **inherit**;  
 }  
 .app\_\_btns>button{  
 *width*: 100%;  
 }  
  
  
}  
  
</style>

*GenreForm.vue :*

<template>  
 <div class**="container"**>  
 <div class**="form"**>  
 <div v-for**="**genre in genres**"** :key**="**genre.id**"** class**="d-flex align-items-center inputGroup"**>  
 <input :id**="**genre.id**"** :value**="**genre.name**"** name**="genre"** v-model**="**genreList**"** type**="checkbox"**/>  
 <label :for**="**genre.id**"**>**{{genre.name}}**</label>  
 </div>  
 <div  
 v-intersection**="**getGenres**"** class**="observer"** >  
 </div>  
 </div>  
 <div class**="d-flex flex-column"**>  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button v-else class**="mt-3 w-100"** @click**="**action(genreList)**"**>**Save**</my-button>  
 </div>  
 </div>  
</template>  
  
<script>  
import {*mapActions*, *mapState*} from "vuex";  
  
export default {  
 name: "GenreForm",  
 mounted(){  
 this.getGenres()  
 this.genreList = this.book\_genres.map(*genre* => *genre*.name)  
 },  
 data(){  
 return{  
 genreList: []  
 }  
 },  
 computed: {  
 ...*mapState*({  
 genres: *state* => *state*.genre.genres,  
 book\_genres: *state* => *state*.book.book.genres,  
 isLoading: *state* => *state*.book.isLoading  
 })  
 },  
 methods: {  
 ...*mapActions*({  
 getGenres: 'genre/getGenres',  
 changeGenres: 'book/changeGenres'  
 }),  
 async action(*list*){  
 const dialog = *document*.querySelector('.dialog')  
 await this.changeGenres(*list*)  
 this.$swal({  
 html: "<span style='color: #ffffff'>The genres have been successfully changed</span>",  
 title: '<span style=\'color: #ffffff\'>Success</span>',  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 }  
  
 }  
  
  
}  
</script>

<style scoped>  
.inputGroup {  
 *background-color*: #fff;  
 *margin*: 3**px** 0;  
 *position*: **relative**;  
 *padding*: 0;  
}  
  
label {  
 *padding*: 12**px** 30**px**;  
 *width*: 50**vw**;  
 *min-width*: 320**px**;  
 *text-align*: **left**;  
 *color*: #3C454C;  
 *cursor*: **pointer**;  
 *position*: **relative**;  
 *z-index*: 2;  
 *transition*: **color** 200**ms ease-in**;  
 *overflow*: **hidden**;  
}  
  
label:after {  
 *width*: 10**px**;  
 *height*: 10**px**;  
 *border-radius*: 50%;  
 *content*: '';  
 *background-color*: #7072F7;  
 *position*: **absolute**;  
 *left*: 50%;  
 *top*: 50%;  
 *transform*: translate(-50%, -50%) scale3d(1, 1, 1);  
 *transition*: **all** 300**ms** cubic-bezier(0.4, 0.0, 0.2, 1);  
 *opacity*: 0;  
 *z-index*: -1;  
}  
  
label:before {  
 *width*: 32**px**;  
 *height*: 32**px**;  
 *content*: '';  
 *background-image*: url("data:image/svg+xml,%3Csvg width='32' height='32' viewBox='0 0 32 32' xmlns='http://www.w3.org/2000/svg'%3E%3Cpath d='M5.414 11L4 12.414l5.414 5.414L20.828 6.414 19.414 5l-10 10z' fill='%23fff' fill-rule='nonzero'/%3E%3C/svg%3E ");  
 *background-repeat*: **no-repeat**;  
 *background-position*: 4**px** 5**px**;  
 *border-radius*: 50%;  
 *z-index*: 2;  
 *position*: **absolute**;  
 *right*: 20**px**;  
 *top*: 50%;  
 *transform*: translateY(-50%);  
 *cursor*: **pointer**;  
 *transition*: **all** 200**ms ease-in**;  
}  
  
  
input:checked ~ label {  
 *color*: #fff;  
 /\*width: 100%;\*/  
}  
input:checked ~ label:after {  
 *transform*: translate(-50%, -50%) scale3d(56, 56, 1);

*opacity*: 1;  
}  
  
input:checked ~ label:before {  
 *background-color*: #54E0C7;  
 *border*: **none**;  
}  
  
input {  
 *width*: 0;  
 *height*: 0;  
 *order*: 1;  
 *z-index*: 2;  
 *right*: 30**px**;  
 *top*: 50%;  
 *transform*: translateY(-50%);  
 *cursor*: **pointer**;  
 *visibility*: **hidden**;  
}  
.form {  
 *max-width*: 550**px**;  
 *overflow-y*: **auto**;  
 *overflow-x*: **hidden**;  
 *max-height*: 500**px**;  
  
}  
  
code {  
 *background-color*: #9AA3AC;  
 *padding*: 0 8**px**;  
}  
  
.container{  
 *min-width*: **max-content**;  
}  
  
@media **screen** and (*max-width*: 750**px**){  
 label{  
 *width*: 90**vw**;  
 }  
}  
  
</style>

*GenreList.vue :*

<template>  
 <div>  
 <table class**="genre\_\_table"**>  
 <caption>**Genres :** </caption>  
 <thead>  
 <tr>  
 <th>**Name**</th>  
 <th></th>  
 </tr>  
 </thead>  
 <tbody>  
 <transition-group name**="list"**>  
 <tr v-for**="**genre in genres**"** :key**="**genre.id**"**>  
 <td>**{{genre.name}}**</td>  
 <td>  
 <div class**="d-flex justify-content-center gap-3"**>  
 <my-button @click**="**showGenreUpdateDialog(genre)**"**>**Edit**</my-button>  
 <my-button @click.once**="**removeGenre(genre.id)**"**> **Delete** </my-button>  
 </div>  
 </td>  
 </tr>  
 </transition-group>  
 <tr>  
 <td>  
 <div  
 v-intersection**="**getGenres**"** class**="observer"** >  
 </div>  
 </td>  
 </tr>  
 </tbody>  
 </table>  
 <my-dialog v-model:show**="**genreUpdateDialogVisible**"**>  
 <create-genre-form :modified**="**true**"**>  
 <template v-slot:submit\_\_name>  
 **Save** </template>  
 </create-genre-form>  
 </my-dialog>  
 </div>  
  
</template>  
  
<script>  
  
import {*mapActions*, *mapState*} from "vuex";  
import MyButton from "./UI/MyButton";  
import CreateGenreForm from "./CreateGenreForm";  
import MyDialog from "./UI/MyDialog";  
  
export default {  
 name: "GenreList",  
 components: {CreateGenreForm, MyButton, MyDialog},  
 data(){  
 return{  
 genreUpdateDialogVisible: false  
 }  
 },  
 computed:{  
 ...*mapState*({  
 user: *state* => *state*.user.user,  
 genres: *state* => *state*.genre.genres  
 }),  
 },  
 methods: {  
 ...*mapActions*({  
 removeGenre: 'genre/removeGenre',  
 getGenres: 'genre/getGenres'  
 }),  
 showGenreUpdateDialog(*genre*){  
 this.genreUpdateDialogVisible = true  
 this.$store.commit('genre/setGenre', *genre*)  
 }  
  
 },  
}  
</script>

<style scoped>  
caption {  
 *background*: rgba(56, 58, 73, 0.9);  
 *color*: **azure**;  
 *padding*: 10**px**;  
 *text-align*: **left**;  
 *font-size*: 20**px**;  
 *caption-side*: **top**;  
}  
th {  
 *border-bottom*: 3**px solid** rgba(56, 58, 73, 0.9);  
 *padding*: 10**px**;  
 *text-align*: **left**;  
}  
td {  
 *padding*: 10**px**;  
}  
tr{  
 *min-width*: 100%;  
}  
tr:nth-child(odd) {  
 *background*: rgba(72, 86, 95, 0.2);  
}  
tr:nth-child(even) {  
 *background-color*: rgba(56, 58, 73, 0.9);  
}  
  
tbody{  
 *display*: **block**;  
 *overflow-y*: **auto**;  
 *max-height*: 500**px**;  
}  
  
table thead, table tbody tr {  
 *display*: **table**;  
 *width*: 100%;  
 *table-layout*: **fixed**;  
}  
  
  
</style>

*OrderForm.vue :*

<template>  
 <Form v-slot**="**{ *handleSubmit* }**"** :validation-schema**="**schema**"** as**="div"** class**="order\_\_form"**>  
 <my-error-list :errors**="**errors**"**></my-error-list>  
 <form @submit**="***handleSubmit*(*$event*, action)**"** method**="post"** class**="d-flex flex-column form"** >  
 **Country :** <my-field  
 v-model**="**order.country**"** name**="country"** />  
 <my-error-message name**="country"** />  
 **City :** <my-field  
 v-model**="**order.city**"** name**="city"** />  
 <my-error-message name**="city"** />  
 **Address :** <my-field  
 v-model**="**order.address**"** name**="address"** />  
 <my-error-message name**="address"** />  
 **Delivery date:** <my-field  
 v-model**="**order.deliver\_date**"** type**="date"** name**="deliver"** />  
 <my-error-message name**="deliver"** />  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center m-2"**></div>  
 <my-button  
 v-else  
 type**="submit"** >  
 <slot name**="submit\_\_name"**></slot>  
 </my-button>  
 </form>  
 </Form>  
</template>  
  
<script>  
import {*mapActions*, *mapMutations*, *mapState*} from "vuex";  
import {*Form*} from 'vee-validate'  
  
import \* as yup from 'yup'  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import MyErrorList from "./UI/MyErrorList";  
import *router* from "../router/router";  
  
  
export default {  
 name: "OrderForm",  
 components: {*Form*, MyField, MyErrorList, MyErrorMessage},  
 props: {  
 modified:{  
 type: *Boolean*,  
 default: false  
 },  
 },  
 mounted() {  
 this.clearErrors()  
 },  
 methods: {  
 ...*mapMutations*({  
 clearErrors: 'clearErrors'  
 }),  
 ...*mapActions*({  
 createOrder: 'order/createOrder',  
 }),  
 async action(){  
 const dialog = *document*.querySelector('.dialog')  
 await this.createOrder()  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>Order successfully created</span>",  
 icon: 'success',  
 showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 },  
 },  
 computed: {

...*mapState*({  
 order: *state* => *state*.order.order,  
 errors: *state* => *state*.errors,  
 isLoading: *state* => *state*.order.isLoading  
 }),  
 schema() {  
 return yup.object().shape({  
 country: yup.string().required().label('Country'),  
 city: yup.string().required().label('City'),  
 address: yup.string().required().label('Address'),  
 deliver: yup.date().min(new *Date*(), 'Delivery date must be later than today').required().label('Delivery date'),  
 })  
 },  
 },  
}  
</script>  
  
<style scoped>  
.order\_\_form{  
 *width*: 25**vw**;  
 *min-width*: 200**px**;  
 *max-width*: 500**px**;  
}  
</style>

*OrderList.vue :*

<template>  
 <div>  
 <table class**="\_table"**>  
 <caption>  
 <slot name**="image"**></slot>  
 **Orders :** </caption>  
 <thead>  
 <tr>  
 <th v-if**="**isAdmin**"**>**User**</th>  
 <th>**Delivery date**</th>  
 <th>**Destination**</th>  
 <th>**Books**</th>  
 <th></th>  
 </tr>  
 </thead>  
 <tbody>  
 <transition-group name**="list"**>  
 <tr v-for**="**order in orders**"** :key**="**order.id**"** >  
 <td v-if**="**isAdmin**"**>**{{order.user\_id}}**</td>  
 <td>**{{new Date(order.deliver\_date).toLocaleDateString()}}**</td>  
 <td>**{{order.country}}, {{order.city}}, {{order.address}}**</td>  
 <td class**="d-flex flex-wrap gap-2"**>  
 <router-link v-for**="**book in order.books**"** :key**="**book**"** class**="text-decoration-none order p-1"** :to**="**`/books/${book.book\_id}`**"**>  
 <span>**Count : {{book.order\_count}}**</span>  
 </router-link>  
 </td>  
 <td>  
 <div class**="btns"**>  
 <my-button @click.once**="**removeOrder(order.id)**"**> **Delete** </my-button>  
 </div>  
 </td>  
 </tr>  
 </transition-group>  
 <tr>  
 <td>  
 <div  
 v-intersection:[isAdmin]**="**getOrders**"** class**="observer"** >  
 </div>  
 </td>  
 </tr>  
 </tbody>  
 </table>  
  
 </div>  
</template>  
  
<script>  
  
import {*mapActions*, *mapState*} from "vuex";  
import MyButton from "./UI/MyButton";  
import UserForm from "./UserForm";  
import MyDialog from "./UI/MyDialog";  
  
export default {  
 name: "OrderList",  
 components: {MyButton, MyDialog},  
 props: {  
 isAdmin: {  
 type: *Boolean*,  
 default: false  
 }  
 },  
 computed:{  
 ...*mapState*({  
 order: *state* => *state*.order.order,  
 orders: *state* => *state*.order.orders,  
 email: *state* => *state*.user.user.email,  
 }),  
  
 },  
 methods: {  
 ...*mapActions*({  
 removeOrder: 'order/removeOrder',  
 getOrders: 'order/getOrders'  
 }),  
 },  
}  
</script>  
  
<style scoped>  
caption {  
 *background*: rgba(56, 58, 73, 0.9);  
 *color*: **azure**;  
 *padding*: 10**px**;  
 *text-align*: **left**;  
 *font-size*: 20**px**;  
 *caption-side*: **top**;  
}  
th {  
 *border-bottom*: 3**px solid** rgba(56, 58, 73, 0.9);  
 *padding*: 10**px**;  
 *text-align*: **left**;

}  
td {  
 *padding*: 10**px**;  
}  
tr{  
 *min-width*: 100%;  
}  
tr:nth-child(odd) {  
 *background*: rgba(72, 86, 95, 0.2);  
}  
tr:nth-child(even) {  
 *background-color*: rgba(56, 58, 73, 0.9);  
}  
.btns{  
 *display*: **flex**;  
 *gap*: 2**px**;  
 *justify-content*: **space-evenly**;  
}  
  
tbody{  
 *display*: **block**;  
 *overflow-y*: **auto**;  
 *height*: 70**vh**;  
}  
  
table thead, table tbody tr {  
 *display*: **table**;  
 *width*: 100%;  
 *table-layout*: **fixed**;  
}  
  
.order{  
 *background-color*: rgba(112, 114, 247, 0.3);  
 *border*: 1**px solid** rgba(112, 114, 247, 0.5);  
 *border-radius*: 5**px**;  
 *margin*: 0;  
 *white-space*: **nowrap**;  
 *text-decoration*: **none**;  
}  
  
</style>

*UserForm.vue :*

<template>  
 <Form v-slot**="**{ handleSubmit }**"** :validation-schema**="**schema**"** as**="div"** class**="user\_\_form"**>  
 <my-title><slot name**="header"**></slot></my-title>  
 <my-error-list :errors**="**errors**"**></my-error-list>  
 <form @submit**="**handleSubmit($event, action)**"** method**="post"** class**="form"**>  
 **Email :** <my-field  
 v-model**="**user.email**"** name**="email"** :disabled**="**this.modified**"** />  
 <my-error-message name**="email"** />  
 **Username :** <my-field  
 v-model**="**user.username**"** name**="username"** />  
 <my-error-message name**="username"** />  
 **First name :** <my-field  
 v-model**="**user.first\_name**"** name**="firstname"** />  
 <my-error-message name**="firstname"** />  
 **Last name :** <my-field  
 v-model**="**user.last\_name**"** name**="lastname"** />  
 <my-error-message name**="lastname"** />  
 <slot name**="password"**></slot> **:** <my-field  
 v-model**="**user.password**"** name**="password"** type**="password"** />  
 <my-error-message name**="password"** />  
 <my-button  
 type**="submit"** >  
 <slot name**="submit\_\_name"**></slot>  
 </my-button>  
 <slot name**="link"**></slot>  
 </form>  
 </Form>  
</template>  
  
<script>  
import {mapActions, mapState} from "vuex";  
import {Form} from 'vee-validate'  
  
import \* as yup from 'yup'  
import MyField from "@/components/UI/MyField";  
import MyErrorMessage from "@/components/UI/MyErrorMessage";  
import MyErrorList from "./UI/MyErrorList";  
import MyTitle from "./UI/MyTitle";  
  
export default {  
 name: "UserForm",  
 components: {  
 MyTitle,  
 MyErrorList,  
 Form, MyField, MyErrorMessage  
 },  
 props: {  
 modified:{  
 type: Boolean,  
 default: false  
 }  
 },  
 methods: {  
 ...mapActions({  
 register: 'user/register',  
 updateUser: 'user/updateUser'  
 }),  
 async action(){  
 const dialog = document.querySelector('.dialog')  
 if(this.modified) {  
 this.updateUser()  
 if(this.errors.length === 0) {  
 this.$swal({  
 title: "<span style='color: #ffffff'>Success</span>",  
 html: "<span style='color: #ffffff'>Account updated successfully</span>",  
 icon: 'success',

showConfirmButton: false,  
 timer: 1000  
 })  
 dialog.click()  
 }  
 }  
 else {  
 this.register()  
 }  
 }  
  
 },  
 computed: {  
 ...mapState({  
 user: state => state.user.user,  
 errors: state => state.errors  
 }),  
 schema() {  
 let schema = yup.object().shape({  
 email: yup.string().email().max(50).required().label('Email'),  
 username: yup.string().max(50).required().label('Username'),  
 firstname: yup.string().max(50).nullable().label('First name'),  
 lastname: yup.string().max(50).nullable().label('Last name'),  
 })  
  
 let password  
 if(!this.modified)  
 password = {password: yup.string().min(5).required().label('Password')}  
 else  
 password = {password: yup.string().min(5).label('Password')}  
 return schema.shape(password)  
 },  
 },  
}  
</script>  
  
<style scoped>  
  
</style>

*UserList.vue :*

<template>  
 <div>  
 <table class**="\_table"**>  
 <caption>**Users :** </caption>  
 <thead>  
 <tr>  
 <th>**Username**</th>  
 <th>**Email**</th>  
 <th>**First Name**</th>  
 <th>**Last Name**</th>  
 <th>**Admin**</th>  
 <th></th>  
 </tr>  
 </thead>  
 <tbody>  
 <transition-group name**="list"**>  
 <tr v-for**="**user in users**"** :key**="**user.id**"**>  
 <td>**{{user.username}}**</td>  
 <td>**{{user.email}}**</td>  
 <td>**{{user.first\_name}}**</td>  
 <td>**{{user.last\_name}}**</td>  
 <td>  
 <div class**="d-flex align-items-center"**>  
 <input type**="checkbox"** class**="disabled"** :id**="**user.id**"** v-model**="**user.role**"** @click**="**changeRole({role: !user.role, id: user.id})**"** :checked**="**user.role\_id === 1**"** :disabled**=**current(user.id)  
 />  
 <label :for**="**user.id**"**></label>  
 </div>  
 </td>  
 <td>  
 <div class**="btns"**>  
 <my-button @click**="**showUserUpdateDialog(user)**"**>**Edit**</my-button>  
 <my-button @click.once**="**removeUser(user.id)**"** v-if**="**!current(user.id)**"**> **Delete** </my-button>  
 </div>  
 </td>  
 </tr>  
 </transition-group>  
 <tr>  
 <td>  
 <div  
 v-intersection**="**getUsers**"** class**="observer"** >  
 </div>  
 </td>  
 </tr>  
 </tbody>  
 </table>  
 <my-dialog v-model:show**="**userUpdateDialogVisible**"**>  
 <user-form :modified**="**true**"**>  
 <template v-slot:password>  
 **New password** </template>  
 <template v-slot:submit\_\_name>  
 **Save** </template>  
 </user-form>  
 </my-dialog>  
 </div>  
</template>  
  
<script>  
  
import {mapActions, mapState} from "vuex";  
import MyButton from "./UI/MyButton";  
import UserForm from "./UserForm";  
import MyDialog from "./UI/MyDialog";  
  
export default {  
 name: "UserList",  
 components: {UserForm, MyButton, MyDialog},  
 data(){  
 return{  
 userUpdateDialogVisible: false,  
 }  
 },  
 computed:{  
 ...mapState({  
 user: state => state.user.user,

users: state => state.user.users  
 }),  
  
 },  
 methods: {  
 ...mapActions({  
 removeUser: 'user/removeUser',  
 changeRole: 'user/changeRole',  
 getUsers: 'user/getUsers'  
 }),  
 showUserUpdateDialog(user){  
 this.userUpdateDialogVisible = true  
 this.$store.commit('user/setUser', user)  
 },  
 current(id){return JSON.parse(localStorage.user\_id) === id}  
  
 },  
}  
</script>  
  
<style scoped>  
  
caption {  
 *background*: rgba(56, 58, 73, 0.9);  
 *color*: **azure**;  
 *padding*: 10**px**;  
 *text-align*: **left**;  
 *font-size*: 20**px**;  
 *caption-side*: **top**;  
}  
th {  
 *border-bottom*: 3**px solid** rgba(56, 58, 73, 0.9);  
 *padding*: 10**px**;  
 *text-align*: **left**;  
}  
td {  
 *padding*: 10**px**;  
 *word-break*: **break-word**;  
}  
tr{  
 *min-width*: 100%;  
}  
tr:nth-child(odd) {  
 *background*: rgba(72, 86, 95, 0.2);  
}  
tr:nth-child(even) {  
 *background-color*: rgba(56, 58, 73, 0.9);  
}  
.btns{  
 *display*: **flex**;  
 *gap*: 2**px**;  
 *justify-content*: **space-evenly**;  
}  
  
tbody{  
 *display*: **block**;  
 *overflow-y*: **auto**;  
 *max-height*: 500**px**;  
}  
  
table thead, table tbody tr {  
 *display*: **table**;  
 *width*: 100%;  
 *table-layout*: **fixed**;

}  
  
input[type=checkbox]{  
 *height*: 0;  
 *width*: 0;  
 *visibility*: **hidden**;  
}  
label {  
 *cursor*: **pointer**;  
  
 *width*: 40**px**;  
 *height*: 20**px**;  
 *background*: rgba(149, 149, 149, 0.34);  
 *display*: **block**;  
 *border-radius*: 100**px**;  
 *position*: **relative**;  
}  
label:after {  
 *content*: '';  
 *position*: **absolute**;  
 *top*: 0;  
 *left*: 0;  
 *width*: 20**px**;  
 *height*: 20**px**;  
 *background*: #fff;  
 *border-radius*: 90**px**;  
 *transition*: 0.3**s**;  
}  
  
input:checked + label {  
 *background*: #7072F7;  
}  
  
input:disabled + label{  
 *background*: rgb(255, 25, 25);  
}  
  
input:checked + label:after {  
 *left*: 100%;  
 *transform*: translateX(-100%);  
}  
  
label:active:after {  
 *width*: 70%;  
}  
  
@media(*max-width*: 1200**px**) {  
  
}  
  
</style>

***UI components :***

*Basket.vue :*

<template>  
 <div @click**="**this.$store.commit('basket/setVisibility', false)**"**>  
 <my-dialog v-model:show**="**visible**"** >  
 <div style**="***background-color*: rgb(23, 28, 33); *border-radius*: 5**px**; *width*: **fit-content"** class**="basket-front"**>  
 <div class**="basket"**>  
 <div class**="d-flex justify-content-between ps-3 pe-3 p-2 "**>

<h3 class**="align-items-center text-center m-0 p-1"**>  
 <slot name**="image"**></slot>  
 **Cart** </h3>  
 <button @click**="**this.$store.commit('basket/setVisibility', false)**"** class**="close"**>**X**</button>  
 </div>  
 <hr/>  
 <div class**="basket-body d-flex flex-column"** :class**="**{'justify-content-center' : books.length === 0 || isLoading}**"**>  
 <div v-if**="**isLoading**"** class**="spinner-grow align-self-center jus"**></div>  
 <transition-group name**="list"** v-else>  
 <basket-book v-for**="**book in books**"** :key**="**book.id**"** :order**="**book**"** class**="p-2"**></basket-book>  
 </transition-group>  
 <div v-intersection**="**getBooks**"** class**="observer"**></div>  
 <div v-if**="**books.length === 0 && !isLoading**"** class**="d-flex flex-column justify-content-center align-items-center"**>  
 <img src**="@/assets/box\_negate.png"** height**="100"** width**="100"**/>  
 <span>**Your cart is empty**</span>  
 </div>  
 </div>  
 <div v-if**="**books.length > 0**"** class**="basket-btns"**>  
 <hr/>  
 <div class**="d-flex justify-content-between p-3"**>  
 <my-button @click**="**this.$store.commit('basket/setVisibility', false)**"** class**="h-50 align-self-center"**>**Back to shopping**</my-button>  
 <div class**="d-flex align-items-center justify-content-end align-items-center gap-3 buy-btn p-2"**>  
 <span>**${{getTotalSum.toFixed(2)}}**</span>  
 <my-button @click**="**showCreateOrderDialog**"**>**Checkout**</my-button>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 <my-dialog v-model:show**="**createOrderDialogVisible**"** v-if**="**this.books.length > 0**"**>  
 <order-form>  
 <template v-slot:submit\_\_name>  
 **Checkout** </template>  
 </order-form>  
 </my-dialog>  
 </my-dialog>  
 </div>  
</template>  
  
<script>  
import {mapActions, mapGetters, mapMutations, mapState} from "vuex";  
import BasketBook from "../BasketBook";  
import MyButton from "./MyButton";  
import MyDialog from "./MyDialog";  
import OrderForm from "../OrderForm";  
  
export default {  
 name: "Basket",  
 components: {OrderForm, MyDialog, MyButton, BasketBook},  
 data(){  
 return{  
 createOrderDialogVisible: false  
 }

},  
 updated() {  
 if(this.books.length === 0){  
 this.createOrderDialogVisible = false  
 }  
 },  
 computed: {  
 ...mapState({  
 isAuth: state => state.isAuth,  
 visible: state => state.basket.visible,  
 isLoading: state => state.basket.isLoading,  
 books: state => state.basket.books  
 }),  
 ...mapGetters({  
 getTotalSum: 'basket/getTotalSum'  
 })  
 },  
 methods: {  
 ...mapActions({  
 getBooks: 'basket/getBasketBooks'  
 }),  
 ...mapMutations({  
 clearBasket: 'basket/clearBasket'  
 }),  
 async showCreateOrderDialog(){  
 this.createOrderDialogVisible = true  
 },  
 }  
}  
</script>  
  
<style scoped>  
.basket{  
 *max-width*: 50**vw**;  
 *min-width*: 350**px**;  
 *background-color*: rgba(149, 149, 149, 0.3);  
 *border-radius*: 5**px**;  
 *color*: #292a5c;  
}  
.basket-front{  
 *width*: 105%;  
}  
.basket-body{  
 *overflow-y*: **auto**;  
 *overflow-x*: **hidden**;  
 *height*: 60**vh**;  
 *padding*: 10**px**;  
}  
hr{  
 *margin*: 0;  
}  
.close{  
 *display*: **flex**;  
 *justify-content*: **center**;  
 *align-items*: **center**;  
 *text-align*: **center**;  
 *width*: **fit-content**;  
 *background-color*: **inherit**;  
 *border*: **none**;  
}  
  
  
.close:hover{  
 *color*: **red**;

}  
  
.buy-btn{  
 *background-color*: rgba(112, 114, 247, 0.2);  
 *border-radius*: 5**px**;  
 *border*: **solid** 2**px** rgba(112, 114, 247, 0.4);  
}  
  
@media **screen** and (*max-width*: 1400**px**){  
 .basket-btns>div{  
 *flex-direction*: **column-reverse**;  
 *gap*: 0.5**rem** }  
 .basket-btns button{  
 *width*: 100%;  
 }  
 .buy-btn{  
 *flex-direction*: **column**;  
 }  
}  
</style>

*EmptyList.vue :*

<template>  
 <div class**="empty\_\_list"**>  
 <img src**="@/assets/wrong\_negate.png"** height**="128"** width**="128"**/>  
 <h4>  
 <slot></slot>  
 </h4>  
 </div>  
</template>  
  
<script>  
export default {  
 name: "EmptyList"  
}  
</script>  
  
<style scoped>  
  
.empty\_\_list{  
 *width*: 100%;  
 *min-height*: 80**vh**;  
 *display*: **flex**;  
 *justify-content*: **center**;  
 *flex-direction*: **column**;  
 *align-items*: **center**;  
 *position*: **absolute**;  
}  
</style>

*MyButton.vue :*

<template>  
 <button type**="button"** class**="out-btn p-3"**>  
 <slot></slot>  
 </button>  
</template>  
  
<script>  
export default {  
 name: "my-button"

}

</script>  
  
<style scoped>  
.out-btn{  
 *font-size*: 12**px !important**;  
 *border-radius*: 5**px**;  
 *background-color*: #7072F7;  
 *border*: **none**;  
 *color*: #ffffff;  
 *transition*: **all** 0.3**s ease-in-out**;  
 *white-space*: **nowrap**;  
}  
.out-btn:hover {  
 *background-color*: **white**;  
 *color*: #292a5c;  
}  
.out-btn:active {  
 *background-color*: **inherit !important**;  
 *color*: **white**;  
  
}  
  
</style>

*MyDialog.vue :*

<template>  
 <div class**="dialog"** v-if**="**show**"** @click**="**hideDialog**"** v-focus>  
 <div class**="dialog\_\_content"** @click.stop>  
 <slot></slot>  
 </div>  
 </div>  
</template>  
  
<script>  
  
import toggleDialogMixin from "@/mixins/toggleDialogMixin";  
  
export default {  
 name: "MyDialog",  
 mixins: [toggleDialogMixin],  
}  
</script>  
  
<style scoped>  
.dialog{  
 *top*: 0;  
 *right*: 0;  
 *left*: 0;  
 *bottom*: 0;  
 *background*: rgba(0, 0, 0, 0.7);  
 *position*: **fixed**;  
 *display*: **flex**;  
 *z-index*: 100;  
}  
  
.dialog\_\_content{  
 *margin*: **auto**;  
 *border-radius*: 5**px**;  
}  
</style>

*MyErrorMessage.vue :*

<template>  
 <template v-if**="**errors.length**"**>  
 <li v-for**="**message in errors**"** :key**="**message**"** class**="error"**>**{{ message }}**</li>  
 </template>  
</template>  
  
<script>  
  
export default {  
 name: "MyErrorList",  
 props: {  
 errors: {  
 type: *Array*,  
 required: true  
 }  
 },  
}  
</script>  
  
<style scoped>  
li {  
 *list-style-type*: **none**;  
 *overflow-wrap*: **break-word**;  
}  
</style>

*MyErrorMessage.vue :*

<template>  
 <ErrorMessage class**="error"**/>  
</template>  
  
<script>  
import {*ErrorMessage*} from "vee-validate"  
export default {  
 name: "MeErrorMessage",  
 components:{  
 *ErrorMessage* }  
}  
</script>  
  
<style scoped>  
</style>

*NyField.vue :*

<template>  
 <Field class**="input"**/>  
</template>  
  
<script>  
import {Field} from 'vee-validate'  
export default {  
 name: "My-Field",  
 components: {  
 Field  
 }

}  
</script>  
  
<style scoped>  
  
</style>

*MyInput.vue :*

<template>  
 <input :value**='**modelValue**'** @input**="**updateInput**"** class**="input"**>  
</template>  
  
<script>  
export default {  
 name: "my-input",  
 props: {  
 modelValue: [*String*, *Number*],  
 },  
 methods: {  
 updateInput(*event*){  
 this.$emit('update:modelValue', *event*.target.value)  
 }  
 }  
}  
  
</script>  
  
<style scoped>  
</style>

*MySearch.vue :*

<template>  
 <form @submit.prevent**="**Search**"** class**="search-bar"**>  
 <input  
 type**="search"** name**="search"** v-model**="**query.value**"** placeholder**="Search"** >  
 <button class**="search-btn"** type**="submit"**>  
 <span>**Search**</span>  
 </button>  
 </form>  
</template>  
  
<script>  
import MyInput from "./MyInput";  
import MyButton from "./MyButton";  
import {mapActions, mapMutations, mapState} from "vuex";  
export default {  
 name: "MySearch",  
 components: {MyButton, MyInput},  
 computed: {  
 ...mapState({  
 query: state => state.book.searchQuery,  
 }),  
 },  
 methods: {  
 ...mapActions({  
 getBookList: 'book/getSortedBookList'

}),  
 ...mapMutations({  
 clearBookStore: 'book/clearBookStore',  
 setSearchedMod: 'book/setSearchedMod'  
 }),  
 Search(){  
 this.clearBookStore()  
 this.setSearchedMod(true)  
 this.getBookList()  
 }  
 }  
  
}  
</script>  
  
<style scoped>  
\* {  
 *border*: 0;  
 *box-sizing*: **border-box**;  
 *margin*: 0;  
 *padding*: 0;  
}  
/\*:root {\*/  
/\* font-size: calc(16px + (24 - 16)\*(100vw - 320px)/(1920 - 320));\*/  
/\*}\*/  
body, button, input {  
 *font*: 1**em** Hind, **sans-serif**;  
 *line-height*: 1.5**em**;  
}  
body, input {  
 *color*: #171717;  
}  
body, .search-bar {  
 *display*: **flex**;  
}  
body {  
 *background*: **white**;  
 *height*: 100**vh**;  
}  
.search-bar input,  
.search-btn,  
.search-btn:before,  
.search-btn:after {  
 *transition*: **all** 0.25**s ease-out**;  
}  
.search-bar input,  
.search-btn {  
 *width*: 3**em**;  
 *height*: 3**em**;  
}  
.search-bar input:invalid:not(:focus),  
.search-btn {  
 *cursor*: **pointer**;  
}  
.search-bar,  
.search-bar input:focus,  
.search-bar input:valid {  
 *width*: 100%;  
}  
.search-bar {  
 *margin*: **auto**;  
 *justify-content*: **end**;  
 *max-width*: 30**em**;  
}

.search-bar input {  
 *background*: **transparent**;  
 *border-radius*: 1.5**em**;  
 *box-shadow*: 0 0 0 0.4**em** #171717 **inset**;  
 *padding*: 0.75**em**;  
 *transform*: translate(0.5**em**,0.5**em**) scale(0.5);  
 *transform-origin*: 100% 0;  
 *-webkit-appearance*: **none**;  
 *-moz-appearance*: **none**;  
 *appearance*: **none**;  
}  
.search-bar input::-webkit-search-decoration {  
 *-webkit-appearance*: **none**;  
}  
.search-bar input:focus,  
.search-bar input:valid {  
 *background*: **white**;  
 *border-radius*: 0.375**em** 0 0 0.375**em**;  
 *box-shadow*: 0 0 0 0.1**em** #d9d9d9 **inset**;  
 *transform*: scale(1);  
}  
.search-btn {  
 *background*: #171717;  
 *border-radius*: 0 0.75**em** 0.75**em** 0 / 0 1.5**em** 1.5**em** 0;  
 *padding*: 0.75**em**;  
 *position*: **relative**;  
 *transform*: translate(0.25**em**,0.25**em**) rotate(45**deg**) scale(0.25,0.125);  
 *transform-origin*: 0 50%;  
}  
.search-btn:before,  
.search-btn:after {  
 *content*: "";  
 *display*: **block**;  
 *opacity*: 0;  
 *position*: **absolute**;  
}  
.search-btn:before {  
 *border-radius*: 50%;  
 *box-shadow*: 0 0 0 0.2**em white inset**;  
 *top*: 0.75**em**;  
 *left*: 0.75**em**;  
 *width*: 1.2**em**;  
 *height*: 1.2**em**;  
}  
.search-btn:after {  
 *background*: **white**;  
 *border-radius*: 0 0.25**em** 0.25**em** 0;  
 *top*: 51%;  
 *left*: 55%;  
 *width*: 0.75**em**;  
 *height*: 0.25**em**;  
 *transform*: translate(0.2**em**,0) rotate(45**deg**);  
 *transform-origin*: 0 50%;  
}  
.search-btn span {  
 *display*: **inline-block**;  
 *overflow*: **hidden**;  
 *width*: 1**px**;  
 *height*: 1**px**;  
}  
  
/\* Active state \*/  
.search-bar input:focus + .search-btn,  
.search-bar input:valid + .search-btn {

*background*: #7072F7;  
 *border-radius*: 0 0.375**em** 0.375**em** 0;  
 *transform*: scale(1);  
}  
.search-bar input:focus + .search-btn:before,  
.search-bar input:focus + .search-btn:after,  
.search-bar input:valid + .search-btn:before,  
.search-bar input:valid + .search-btn:after {  
 *opacity*: 1;  
}  
.search-bar input:focus + .search-btn:hover,  
.search-bar input:valid + .search-btn:hover,  
.search-bar input:valid:not(:focus) + .search-btn:focus {  
 *background*: #7072F7;  
}  
.search-bar input:focus + .search-btn:active,  
.search-bar input:valid + .search-btn:active {  
 *transform*: translateY(1**px**);  
}  
  
@media **screen** and (*prefers-color-scheme*: **dark**) {  
 body, input {  
 *color*: **white**;  
 }  
 body {  
 *background*: #171717;  
 }  
 .search-bar input {  
 *box-shadow*: 0 0 0 0.4**em white inset**;  
 }  
 .search-bar input:focus,  
 .search-bar input:valid {  
 *background*: rgba(149, 149, 149, 0.34);  
 *box-shadow*: 0 0 0 0.1**em** #3d3d3d **inset**;  
 }  
 .search-btn {  
 *background*: #f1f1f1;  
 }  
}  
  
</style>

*MySelect.vue :*

<template>  
 <select v-model**="**value**"** @change**="**changeOption**"** class**="search"**>  
 <option value**=""** disabled>**Select from the list**</option>  
 <option  
 v-for**="**option in options**"** :key**="**option.value**"** :value**="**option.value**"** >  
 **{{ option.name }}** </option>  
 </select>  
</template>  
  
<script>  
export default {  
 name: "my-select",  
 props:{  
 modelValue: [*String*],  
 options:{  
 type: *Array*,  
 default: () => []  
 }  
 },  
 data(){  
 return{  
 value: this.modelValue  
 }  
 },  
 methods:{  
 changeOption(*event*){  
 this.$emit('update:modelValue', *event*.target.value)  
 this.$store.commit('book/clearBooks')  
 }  
  
 }  
  
}  
</script>  
  
<style scoped>  
.search{  
 *height*: 30**px**;  
 *color*: #292a5c;  
}  
  
.search option{  
 *color*: #292a5c;  
}  
</style>

*MyTitle.vue :*

<template>  
 <h2 class**="title"**><slot></slot></h2>  
</template>  
  
<script>  
export default {  
 name: "my-title"  
}  
</script>  
  
<style scoped>  
.title{  
 *color*: #7072F7;  
  
}  
</style>

*Navbar.vue :*

<template>  
 <div class**="navbar flex-nowrap d-flex gap-2"**>  
 <div @click**="**goToHome**"**>  
 <div class**="logo"**>**PSALT**<span>**ERIUM**</span>  
 </div>  
 </div>  
 <div class**="navbar\_\_btns"**>  
 <div v-show**="**this.currentRoute() === '/'**"**>  
 <my-search />  
 </div>  
 <my-button class**="p-2"** @click**="**showSidebar**"**>

<img src**="@/assets/menu.png"** height**="20"** width**="20"**/>  
 </my-button>  
 <my-dialog v-model:show**="**sidebarVisible**"**>  
 <div class**="sidebar bg-gradient p-3 gap-2 d-flex flex-column h-100"** @click**="**sidebarVisible = false**"**>  
 <router-link to**="/"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/'}**"**>  
 <img src**="@/assets/shop.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Shop**</span>  
 </router-link>  
 <div v-if**="**!isAuth**"**>  
 <router-link to**="/login"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/login'}**"**>  
 <img src**="@/assets/login.png"** height**="32"** width**="32"**/>  
 </my-button>  
 <span>**Log in**</span>  
 </router-link>  
 <router-link to**="/register"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/register'}**"**>  
 <img src**="@/assets/personal.png"** height**="32"** width**="32"**/>  
 </my-button>  
 <span>**Register**</span>  
 </router-link>  
 </div>  
 <div v-else>  
 <router-link to**="/admin"** v-show**="**isAdmin**"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/admin'}**"**>  
 <img src**="@/assets/settings.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Admin**</span>  
 </router-link>  
 <router-link to**="/my/books"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/my/books'}**"**>  
 <img src**="@/assets/book.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Your books**</span>  
 </router-link>  
 <router-link to**="/orders"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/orders'}**"**>  
 <img src**="@/assets/sent.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Orders**</span>  
 </router-link>  
 <router-link to**=""** @click**="**this.$store.commit('basket/setVisibility', true)**"**>  
 <my-button class**="p-2"**>  
 <img src**="@/assets/cart.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Cart**</span>  
 </router-link>  
 <router-link to**="/account"**>  
 <my-button class**="p-2"** :class**="**{'target':this.currentRoute() === '/account'}**"**>  
 <img src**="@/assets/user.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Account**</span>  
 </router-link>

<router-link to**="/login"** @click**="**this.$store.dispatch('logout')**"**>  
 <my-button  
 class**="p-2"** >  
 <img src**="@/assets/logout.png"** height**="32"** width**="32"** />  
 </my-button>  
 <span>**Log out**</span>  
 </router-link>  
 </div>  
 </div>  
 </my-dialog>  
 </div>  
 <basket>  
 <template v-slot:image>  
 <img src**="@/assets/cart\_negate.png"** height**="32"** width**="32"** class**="mb-2"**/>  
 </template>  
 </basket>  
 </div>  
</template>  
  
<script>  
import {mapActions, mapMutations, mapState} from "vuex";  
export default {  
 name: "Navbar",  
 data(){  
 return{  
 sidebarVisible: false  
 }  
 },  
 computed: {  
 ...mapState({  
 searchQuery: state => state.book.searchQuery,  
 isAdmin: state => state.isAdmin,  
 isAuth: state => state.isAuth,  
 books: state => state.basket.books  
 }),  
 isAuth(){  
 return JSON.parse(this.$store.state.isAuth)  
 },  
 goToHome(){  
 if(this.currentRoute() === '/')  
 this.$router.go()  
 else  
 this.$router.push('/')  
 },  
 },  
 methods: {  
 ...mapActions({  
 search: 'book/search'  
 }),  
 ...mapMutations({  
 setSearchQuery: 'book/setSearchQuery'  
 }),  
 currentRoute(){return this.$router.currentRoute.value.fullPath},  
 showSidebar(){this.sidebarVisible = true}  
 },  
  
}  
</script>  
  
<style scoped>  
.logo{  
 *color*: **white**;

*font-size*: 20**px**;  
 *cursor*: **pointer**;  
 *display*: **flex**;  
}  
.logo>span{  
 *color*: #7072F7;  
 *font-size*: **inherit**;  
}  
.navbar{  
 *height*: 62**px**;  
 *display*: **flex**;  
 *align-items*: **center**;  
 *flex-direction*: **row**;  
 *padding*: 10**px** 15**px**;  
 *background-color*: rgb(23, 28, 33);  
 *box-shadow*: 2**px** 5**px** 5**px** 5**px** rgba(23, 28, 33, 1);  
 *z-index*: 100;  
}  
  
.navbar\_\_btns{  
 *margin-left*: **auto**;  
}  
.navbar\_\_btns, .navbar\_\_btns>div{  
 *display*: **flex**;  
 *gap*: 10**px**;  
}  
my-button{  
 *background-color*: **inherit**;  
}  
  
button.target{  
 *background-color*: **white**;  
 *color*: #292a5c;  
 *padding*: 8**px**;  
}  
  
.sidebar{  
 *background-color*: rgba(56, 58, 73, 0.8);  
 *width*: 250**px**;  
 *height*: 100**vh**;  
 *position*: **absolute**;  
 *right*: 0;  
 *top*: 0;  
}  
.sidebar>div{  
 *display*: **flex**;  
 *flex-direction*: **column**;  
 *gap*: 0.5**rem**;  
  
}  
a{  
 *text-decoration*: **none**;  
 *display*: **flex**;  
 *align-items*: **center**;  
 *gap*: 1**rem**;  
}  
  
span{  
 *font-size*: 14**px**;  
}

</style>

*index.html :*

<!DOCTYPE html>  
<html lang**=""**>  
 <head>  
 <meta charset**="utf-8"**>  
 <meta http-equiv**="X-UA-Compatible"** content**="IE=edge"**>  
 <meta name**="viewport"** content**="width=device-width,initial-scale=1.0"**>  
 <link rel**="icon"** href**="<%= BASE\_URL %>feather.ico"**>  
 <link rel**="preconnect"** href**="https://fonts.googleapis.com"**>  
 <link rel**="preconnect"** href**="https://fonts.gstatic.com"** crossorigin>  
 <link href**="https://fonts.googleapis.com/css2?family=M+PLUS+1:wght@100&family=Merriweather:ital,wght@0,300;0,400;0,700;0,900;1,300;1,400;1,700;1,900&display=swap"** rel**="stylesheet"**>  
 <title>**<%= htmlWebpackPlugin.options.title %>**</title>  
 </head>  
 <body>  
 <noscript>  
 <strong>**We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled. Please enable it to continue.**</strong>  
 </noscript>  
 <div id**="app"**></div>  
 </body>  
</html>

*Vue.config.js :*