Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича
Інститут фізико-технічних та комп'ютерних наук
(повна назва інституту/факультету)
кафедра інформаційних технологій та комп'ютерної фізики
(повна назва кафедри)

Створення веб-додатку "Бібліотека" для оптимізації роботи з каталогами книг

Дипломна робота
Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент 4 курсу, групи 417
спеціальності
126 – інформаційні системи та технології
(шифр і назва спеціальності)
Шевчук А. П. _____
(прізвище та ініціали)
Керівник: доктор фіз.-мат. наук, ст.наук.сп.,
Борча Мар'яна Драгошівна _____
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)
Рецензент_____
_____
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)

До захисту допущено:
Протокол засідання кафедри № 19
від „ 17 " червня 2021 р.
зав. кафедри _____ Борча М.Д.

Чернівці – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Інститут фізико-технічних та комп'ютерних наук
Кафедра інформаційних технологій та комп'ютерної фізики

ЗАТВЕРДЖУЮ
Завідувач кафедри
докт. фіз.-мат. наук, доц.
_____М. Д. Борча
" ___" _____ 2021 р.

СТВОРЕННЯ ВЕБ-ДОДАТКУ "БІБЛІОТЕКА" ДЛЯ ОПТИМІЗАЦІЇ РОБОТИ З КАТАЛОГАМИ КНИГ

ЛИСТ ЗАТВЕРДЖЕННЯ

УЗГОДЖЕНО
Керівник роботи
докт. фіз.-мат. наук, доцент
_____ М.Д. Борча
"___" _____ 2021 р.

Виконавець
студент 4-го курсу
_____ А. П. Шевчук
"___" _____ 2021 р.

2021

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Шевчуку Антону Петровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Створення веб-додатку "Бібліотека" для оптимізації роботи з каталогами книг
керівник роботи *Борча Мар'яна Драгошівна, докт. фіз.-мат. наук, доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від "__"_____202_ року №_____

2. Строк подання студентом проекту (роботи) *27 травня 2021 р.*

3. Вихідні дані до проекту (роботи) *Мета роботи – створення та розробка веб-додатку "Бібліотека" для оптимізації роботи з каталогами книг. Розробку веб-додатку виконати з використанням платформи Node.js У якості веб-фреймворк використати Express.js. Двостороннє зв'язування виконати за допомогою фреймворку Angular. Розробку клієнтської частини веб-додатку здійснити на основі відповідних бібліотек. Дослідити роботу додатку з боку адміністраторів, бібліотекарів та читачів.*
Мова програмування – TypeScript та фреймворк Angular

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) описати принцип роботи веб-додатку "Бібліотека"
2) описати закордонні та українські електронні бібліотеки
3) розробити алгоритм роботи веб-додатку "Бібліотека"
4) створити веб-додатку "Бібліотека"
5) дослідити ефективність розробленої програми

5. Перелік графічного матеріалу
1) Логічна схема бази даних
2) Узагальнена діаграма прецедентів програми
3) Функціонал та інтерфейс користувачів веб-додатку

Студент _____ *Шевчук А.П*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Борча М.Д.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота виконана студентом групи 417 Шевчук Антоном Петровичем. Тема «Створення веб-додатку "Бібліотека" для оптимізації роботи з каталогами книг». Робота направлена на здобуття ступеня бакалавр за спеціальністю 126 «Інформаційні системи та технології».

Метою кваліфікаційної роботи є створення та розробка веб-додатку "Бібліотека" для оптимізації роботи з каталогами книг. Актуальність створеного програмного забезпечення полягає у простоті в використанні та зручності отримання потрібної інформації. Створений програмний продукт дозволяє легко переглядати і віддалено забронювати необхідні книги, забезпечує швидкість обробки даної інформації бібліотекарем, і, відповідно, прискорює процес отримання книг користувачем, наочно відображає заборгованість по книгам, показує різноманітну статистику по книгам, користувачам, датам, заборгованостям. Веб-додаток робить зручним для адміністратора управління бібліотекою, в тому числі формування розкладу, створення нових користувачів, полегшує процес замовлення та отримання книг. Система допомагає моніторити загальний процес організації роботи бібліотеки та бути в курсі стану всіх замовлень користувачами та режиму роботи бібліотекарів.

Розробка була виконана з використанням платформи Node.js разом з програмним каркасом Express.js (TypeScript), фреймворку Angular. Як ДОСКБД було обрано MongoDB.

Робота містить 105 сторінок, 52 рисунки та 18 посилання на літературні джерела.

Ключові слова: Веб-додаток, бібліотека, управління, книги, бронювання.

ABSTRACT

Qualification work was performed by a student of the 417 group Shevchuk Anton. Topic "Creating a web application "Library" to optimize the work with book catalogs". The work is aimed at obtaining a bachelor's degree in specialty 126 "Information Systems and Technologies".

The purpose of the qualification work is to create and develop a web application "Library" to optimize the work with book catalogs.

The relevance of the created software lies in the ease of use and ease of obtaining the necessary information. The created software allows you to easily view and remotely book the necessary books, provides speed of processing this information by the librarian, and, accordingly, speeds up the process of receiving books by the user, clearly displays user debts, shows a variety of statistics on books, users, dates, debts. The web application makes it convenient for the administrator to manage the library, including scheduling, creating new users, and facilitates the process of ordering and receiving books. The system helps to monitor the overall process of organizing the work of the library and to be aware of the status of all user orders and the mode of work of librarians.

The development was performed using the platform Node.js with Express.js (TypeScript), Angular framework. MongoDB was chosen as the DBMS.

Number of pages - 105, figures - 52, sources - 18.

Keywords: web application, library, books, managing, booking.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

JS – JavaScript

БД – База Даних

ДОСКБД – Документо-Орієнтована Система Управління Базами Даних

Рис. – Рисунок

ВСТУП

Актуальність роботи

Процес доступу до книг у бібліотеках, паперових чи електронних, повнота наповнення бібліотечних ресурсів в мережі Інтернет, зручність дистанційного замовлення і отримання книг, не завжди є простим, легким і зрозумілим для кінцевого користувача бібліотеки. Часто виникають складнощі з реєстрацією користувача чи відновленням доступу до аккаунту, рідко можна зустріти відображення статистики по авторам, книгам, користувачам, також, практично немає ресурсів для можливості замовити онлайн паперових книг та отримання готового замовлення в бібліотеці, що є достатньо актуальними питаннями в реаліях сучасного світу. Тому використання новітніх бібліотечно-інформаційних технологій для оптимізації управління бібліотекою, отримання легкого і швидкого доступу до книг, без створення черг і витрачання часу, є пріоритетним питанням в парадигмі задоволення інформаційних потреб користувачів. Саме тому, створення веб-додатку, який би автоматизував управління системою, оптимізував роботу з каталогами книг, та загалом підвищив рівень надання користувачам бібліотечних послуг, є надзвичайно актуальним. До того ж, створення і наповнення веб-додатку “Бібліотека”, як ресурсу до легкого доступу до онлайн-книг чи швидкого замовлення книг у бібліотеці, може стати одним із ефективних засобів підтримки наукових досліджень.

Мета роботи: створення багатокористувацького програмного забезпечення - веб-додатку “Бібліотека”, для полегшення роботи працівників бібліотеки та користувачів.

Об’єктом дослідження є створення веб-додатку, який буде використовуватись для управління бібліотекою.

Предметом дослідження є способи проектування та розробки веб-додатку для оптимізації роботи з каталогами книг, методики отримання статистики, зміни даних, інформації користувачами з різним рівнем доступу.

Методи дослідження. В роботі використано платформу Node.js з використанням програмного каркасу Express.js, використовуючи мову програмування TypeScript та фреймворку Angular. Як ДОСКБД було обрано MongoDB.

Теоретична цінність полягає в аналізі предметної області та проектуванні веб-додатку, виборі сучасних методів його реалізації.

Прикладна значущість розробленого додатку полягає в тому, що його можна практично застосовувати для автоматизації та оптимізації управління бібліотекою, в тому числі для можливості легкого замовлення і швидкого доступу до книг користувачам.

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

Перш ніж розпочати розробку нового веб-додатку програмного забезпечення для оптимізації роботи з каталогами книг, варто ознайомитись з уже існуючим електронними бібліотеками, провести пошук аналогів та визначити особливість своєї розробки. Провівши аналітичний огляд існуючих онлайн систем автоматизованого бібліотечного сервісу традиційних провідних бібліотек світу й України, було сформовано перелік електронних бібліотек, зібрано коротку інформацію щодо їх тематичних напрямів, обсяг доступної інформації.

1.1. Міжнародні університети, інститути, наукові та учбові заклади

1. Cambridge University Press.

Тематичні напрямки: суспільні та точні науки, людство. Присутній доступ на перегляд передплачених журналів за алфавітом, видавництвом, темою, новими виданнями. Присутній складний пошук за багатьма критеріями, пов’язаними логічними зв’язками, є налаштування профілю організації та окремого користувача. Доступ до повних текстів обмежений, тип і умови передплати зазначені для кожного типу видання, після оплати користувач отримує доступ до оплаченого тексту. [7]

2. Oxford Reference online.

Тематичні напрямки: усі галузі науки, мистецтво та архітектура, релігія, карти, довідники та словники. Присутній складний пошук за багатьма критеріями, пов’язаними логічними зв’язками з можливістю задання фільтрів, формату виведення результату і обмеження області пошуку. Надають розгалужений спектр доступу до способів оформлення передплати. [13]

3. Electronic Journals Library.

Тематичні напрямки: усі галузі науки, класичні, англійські, американські, романські, слов’янські вчення, природа науки, дослідження, системи вищої освіти, музеї. Здійснюється пошук журналів за різними

критеріями. Є функціонал замовлення онлайн книг і отримання їх у бібліотеці. Для замовлення книг потрібно оформити читацький квиток. [17]

4. SunSite.

Тематичні напрямки: деякі наукові напрямки, домашнє господарство, спорт, дозвілля, мистецтво. Здійснюється пошук додатково по перелікам каталогів, індексів, колекцій. Вільний доступ до більшості інформаційних ресурсів, включаючи повні тексти. [16]

5. Каталог відкритих періодичних видань DOAJ.

Тематичні напрямки: усі галузі науки, релігія, мистецтво, архітектура. Здійснюється пошук за ключовими словами, автором чи словами, назвою, а також словами з анотації. Повністю вільний доступ до всіх інформаційних ресурсів, включаючи повні тексти. [9]

6. ELSEVIER.

Тематичні напрямки: усі галузі науки, техніки, медицини. Присутня фільтрація ненаукових ресурсів при пошуку, є пошук по ключовим словам, по кількості сторінок. Надають вільний доступ до всіх інформаційних ресурсів, включаючи повні тексти. [14]

1.2. Електронні бібліотеки України

В Україні роботи з формування електронних бібліотек почали активно здійснюватись лише останні десятиріччя. Одним з перших кроків у цьому напрямі стало організація передачі копій електронного каталогу Наукової бібліотеки України ім. В.Вернадського в науково-дослідні установи НАН України. Ще однією ініціативою було створення Національної системи електронного інформаційно-бібліотечного ресурсу. [12]

Також, у 2009 році було ухвалено створення єдиної інформаційної бібліотечної системи «Бібліотека–XXI», метою якої стало «підвищення ефективності використання, забезпечення доступності документів, які зберігаються у бібліотечних, архівних та музейних фондах». Повний текст

Розпорядження Кабінету Міністрів та Концепції на веб-сайті Національної бібліотеки України ім. В. І. Вернадського. [5, 6]

На сьогоднішній день електронні бібліотеки знаходяться в процесі постійного розвитку, технічного удосконалення для забезпечення можливостей для користувачів оперативно отримувати необхідну інформацію з будь-якої частини світу.

Розглянемо деякі електронні бібліотеки України.

1. Національна бібліотека імені В.І.Вернадського.

Містить електронні, наукові, інформаційно-аналітичні, історико-культурні, бібліографічні ресурси. Здійснюється пошук в тому числі за ключовими словами. Можливості онлайн-запису у бібліотеку чи бронювання книги онлайн немає. [3]

2. Національна бібліотека України імені Ярослава Мудрого.

До неї належить електронна бібліотека “Культура України” та відкрита електронна бібліотека з деяких наукових напрямків, прикладних наук, мистецтва, дозвілля. Пошукова форма не розширена. Є каталог рідкісних і цінних видань. [4]

3. Бібліотека Верховної Ради України.

Електронна бібліотека включає в себе тексти законів, енциклопедії, словники та довідники. Частина документів зберігається на CD-ROM і не має електронних копій. Наявна можливість замовляти документи через мережу Інтернет. Є розширений перелік полів для пошуку. [1]

4. Наукова бібліотека Національного університету "Києво-Могилянська академія".

Тематичні напрямки: основні галузі науки. Здійснюється розширений пошук. Сторонні користувачі не можуть отримати віддалений доступ до певних електронних ресурсів бібліотеки. Доступ до книг можуть отримати тільки зареєстровані користувачі. Можна замовити онлайн книгу для отримання в бібліотеці.

Основними недоліками в процесі користування даними електронними бібліотеками можна назвати такі: не сучасний дизайн, незручний і незрозумілий пошук, тривалий час завантаження сторінки, обмежений доступ до ресурсів. [2]

1.3. Постановка задачі та область застосування створеного додатку

Веб-додаток “Бібліотека” створений для автоматизації та оптимізації управління бібліотекою, в тому числі для комфортного досвіду користування електронною бібліотекою користувачами, щоб отримати зручний пошук потрібної книги, зрозумілий і простий інтерфейс, легке дистанційне замовлення і швидке отримання книги. Основною метою розробки є створення такого веб-додатку, який буде зручним і простим у користуванні як для бібліотекарів, так і для користувачів. Ціль – створити таке програмне забезпечення, яке автоматизує управління бібліотекою, оптимізує роботу з каталогами книг, та загалом підвищить рівень надання користувачам бібліотечних послуг. Головна ідея полягає не лише у створенні онлайн-ресурсу, що полегшив би управління бібліотекою, а і такого, що дасть позитивний досвід користувачу завдяки інтуїтивно зрозумілому інтерфейсу, отриманню легкого і швидкого доступу до книг, без створення черг і витрачання часу.

При розробці було враховано і виправлено основні недоліки, які існують в процесі користування багатьма електронними бібліотеками. Розроблений веб-додаток є зручним, зрозумілим, швидким, легким у користуванні. Передбачається підтримка різних типів користувачів: адміністраторів, бібліотекарів, користувачів. На сайті є можливість переглядати і віддалено забронювати необхідні книги, переглядати різноманітну статистику по книгам, користувачам, датам, заборгованостям, стану всіх замовлень, формувати розклад, дивитись режим роботи бібліотекарів, створювати нових користувачів, моніторити загальний процес організації роботи бібліотеки.

Висновки до розділу 1

У першому розділі розглянуто різноманітні аналоги розробленого програмного продукту, електронної бібліотеки, для оптимізації роботи з каталогами книг. Розглянуто основні характеристики та визначено загальні недоліки даних аналогів.

Було описано доцільність та мету даної розробки, що дозволяє автоматизувати управління бібліотекою, покращити рівень надання користувачам бібліотечних послуг.

2. ПОБУДОВА СИСТЕМИ

2.1. Платформа Node.js з використанням каркасу Express.js

Node.js - це програмна платформа, основана на двигуні V8 (здійснює трансляцію JS у машинний код), яка перетворює JS із вузькоспеціалізованої мови програмування у мову програмування загального призначення. За допомогою своєї API (написаній на C++) Node.js додаює можливість взаємодіяти із пристроями вводу-виводу, підключати зовнішні бібліотеки, забезпечувати виклики до них з допомогою JavaScript.

Express.js, або просто Express - веб-фреймворк написаний на JavaScript, являється стандартним каркасом для платформи Node.js. Він є мінімалістичним, включає велику кількість підключених плагінів. Express спроектований для створення веб-додатків і API.

Переваги Node.js:

1. Швидкість та продуктивність.
2. Легко маштабується для сучасних додатків.
3. Економічно ефективний з Fullstack JS.
4. Легкий для вивчення і швидкий для адаптування.
5. Покращує час відгуку додатків та підвищує продуктивність.
6. Скорочує час завантаження за допомогою швидкого кешування.
7. Допомагає у створення крос-платформних додатків.
8. Величезна кількість безкоштовних інструментів.
9. Багата екосистема.

Переваги Express.js:

1. Робить розробку веб-додатків Node.js швидкою та простою.
2. Легкий для налаштування і конфігурації.
3. Дозволяє легко визначати маршрути програми на основі методів HTTP та URL-адрес.

4. Включає різні модулі проміжного програмного забезпечення, які можна використовувати для виконання додаткових завдань на ‘запит’ на ‘відповідь’.
5. Дозволяє визначити проміжне середовище для оброблення помилки.
6. Легко обслуговує статичні файли та ресурси програми.
7. Легко підключається до баз даних.
8. Швидко масштабує додаток.
9. Підтримка громади.

В розробці серверної частини додатка для управління бібліотекою були використані такі бібліотеки:

1. @google-cloud/storage - для роботи з на Google Storage: завантаження файлів, отримання доступу до файлів та інше.
2. body-parser - проміжне програмне забезпечення для опрацювання тіла вхідного потоку запитів.
3. bunyan - проста і швидка бібліотека логування.
4. moment - бібліотека JavaScript для перетворення, перевірки, обробки та форматування дат.
5. mongoose - ODM-бібліотека (Object Data Modelling) для роботи з MongoDB. Mongoose працює подібно інструментам ORM.
6. multer - це проміжне програмне забезпечення node.js для обробки multipart/form-data, яке в основному використовується для завантаження файлів.
7. passport - це проміжне програмне забезпечення для автентифікації. Надзвичайно гнучкий та модульний, Passport можна запустити в будь-якій веб-програмі на базі Express. Комплексний набір стратегій підтримує автентифікацію за допомогою імені користувача та пароля, Facebook, Twitter тощо.
8. uuid - для створення UUIDs (Universally Unique IDentifier). UUID має довжину 128 біт і може гарантувати унікальність у просторі та часі.

9. bcryptjs - бібліотека для хешування паролів за допомогою bcrypt. bcrypt є адаптивною криптографічною функцією формування ключа, яка використовується для безпечного зберігання паролів.
10. nodemailer - це модуль для додатків Node.js, який дозволяє легко надсилати електронні листи.
11. base64-img - для перетворення зображень в формат base64 та навпаки із формату base64 в формати jpeg, png та ін.
12. dotenv - для використання змінних середовища.
13. jsonwebtoken - для роботи з jwt токенами. [10, 11]

2.2. Фреймворк Angular

Angular - фреймворк від компанії Google для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків.

Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі. Як головну архітектурну концепцію він застосовує ієрархію компонентів.

Однією з ключових особливостей Angular є те, що він використовує в якості мови програмування TypeScript.

Переваги Angular:

1. Потужна екосистема.
2. Продуктивінсть.
3. Модульна структура розробки.
4. Компонентна архітектура, що забезпечує вищу якість коду.
5. RxJS: efficient, asynchronous programming.
6. Довгострокова підтримка Google.
7. Швидкі оновлення за допомогою Angular CLI.
8. Швидкий процес розвитку.

В розробці клієнтської частини додатка для управління бібліотекою були використані такі бібліотеки:

1. @angular/cli - це інструмент інтерфейсу командного рядка, який використовується для ініціалізації, розробки, побудови ескізів та обслуговування програм Angular безпосередньо з командного терміналу.
2. @angular/forms - інтструмент для роботи з формами.
3. @ngxs/store - це шаблон та бібліотека для управління станом. Він використовується для управління стану всього веб-додатку чи конкретних модулів.
4. @swimlane/ngx-charts - фреймворк для побудови графіків для відображення статистики.
5. moment - бібліотека JavaScript для перетворення, перевірки, обробки та форматування дат.
6. angular-calendar - ангуляр компонент для відображення подій в календарі.
7. bootstrap - безкоштовний набір інструментів з відкритим кодом для створення веб-додатків та веб-сайтів. Він містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript.
8. date-fns - сучасна бібліотека утиліт, яка надає найбільший набір інструментів, але у той же час простий і послідовний, для обробки дат.
9. ngx-image-cropper - утиліта для обрізання, масштабування зображень.
10. RxJS - бібліотека реактивних розширень для JS. Вона надає один основний тип - Observable, супутникові типи (Observer, Schedulers, Subjects) та оператори, натхненні додатками Array # (map, filter, reduce, every тощо), що дозволяють обробляти асинхронні події як колекції. [8]

2.3. Документо-орієнтована система керування базами даних MongoDB

MongoDB - це база даних документів, що означає, що вона зберігає дані у схожих на JSON документах. Також підтримує ad-hoc-запити: вони можуть повертати конкретні поля документів і призначені для користувача JavaScript-функції. Підтримується пошук за регулярними виразами. Також можна налаштувати запит на повернення випадкового набору результатів. MongoDB підтримує індекси (для підвищення продуктивності пошуку в БД).

Система масштабується горизонтально, використовуючи техніку сегментування (англ. Sharding) об'єктів баз даних - розподіл їх частин з різних вузлів кластера. Адміністратор вибирає ключ сегментування, який визначає, за яким критерієм дані будуть рознесені по вузлах (в залежності від значень хеша ключа сегментування). Завдяки тому, що кожен вузол кластера може приймати запити, забезпечується балансування навантаження.

Основним інтерфейсом до бази даних була командна оболонка mongo. З версії MongoDB 3.2 в якості графічної оболонки поставляється «MongoDB Compass». Також існують інші сторонні продукти з своєю графічною оболонкою для адміністрування і перегляду інформації.

Переваги MongoDB:

1. Гнучкі схеми документів.
2. Потужні запити та аналітика.
3. Просте горизонтальне масштабування.
4. Відсутність схеми.
5. Чітка структура окремого об’єкта.
6. Жодних складних об’єднань (SQL Joins).
7. Обробляє великі обсяги даних на високій швидкості за допомогою масштабованої архітектури.
8. Зберігає неструктуровані, напівструктуровані або структуровані дані. [15]

2.4. TypeScript

TypeScript - мова програмування, яка була представлена Microsoft у 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript.

TypeScript є повністю сумісним з мовою програмування JavaScript і компілюється у неї. Після компіляції будь-яку програму, яка написана на TypeScript, можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js. Код експериментального компілятора, який транслює TypeScript в JavaScript, поширюється під ліцензією Apache. Його розробка ведеться в публічному репозиторії через сервіс GitHub. З кожним роком TypeScript набуває все більшої популярності серед мов програмування (рис. 2.1).

TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах), а також підтримкою підключення модулів, що покликане підвищити швидкість розробки, полегшити читаність, рефакторинг і повторне використання коду, допомогти здійснювати пошук помилок на етапі розробки і компіляції, і, можливо, прискорити виконання програм.

TypeScript виник через передбачувані недоліки JavaScript у великомасштабних додатках як в компанії Microsoft, так і у інших користувачів JavaScript. Проблеми з розробкою складних програм на JavaScript призвели до необхідності полегшення розробки компонентів мови

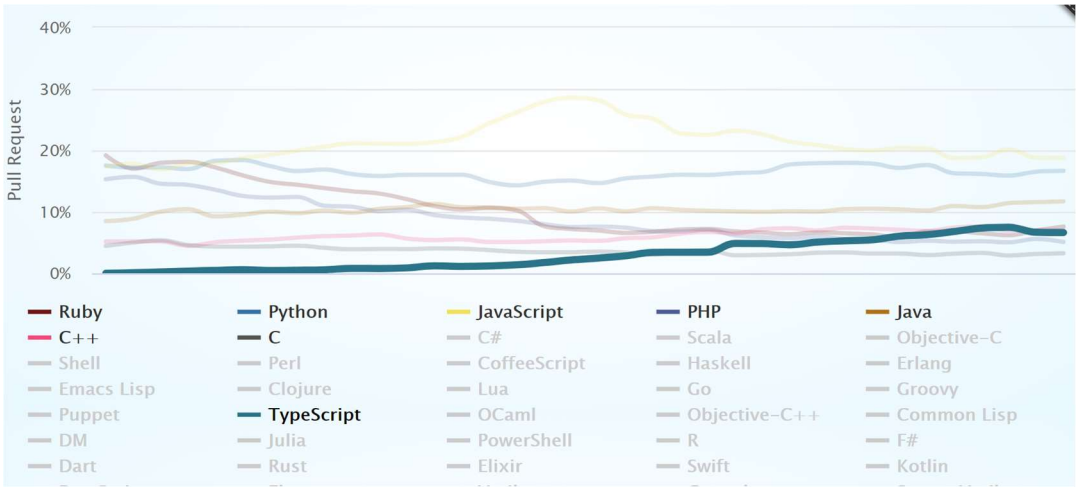


Рис. 2.1. Графік популярності мови програмування TypeScript по роках

Висновки до розділу 2

В даному розділі описується мова та технології програмування. Подано загальну інформацію про мову програмування TypeScript, платформу Node.js із каркасом Express.js, фреймворк Angular, базу даних MongoDB, які використовувався для розробки даного веб-додатку.

Express.js налаштовує зв’язок з базою даних та дозволяє створити REST API для клієнтської частини. Клієнтська частина реалізована з допомогою фреймворку Angular. За основу як серверної так і клієнтської частини взята типізована мова програмування TypeScript.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Загальні відомості та алгоритм веб-додатку

В основу програми з планування масових заходів покладено такий алгоритм роботи:

1. Користувач має можливість зареєструватись в програмі.
   1.1. Користувач повинен перейти на сторінку реєстрації.
   1.2. Користувач повинен заповнити форму реєстрації.
   1.3. Після заповнення реєстрації користувач отримує лист. Користувач повинен перейти по посиланню, яке він отримав у листі для активації акаунту.
2. Користувач має можливість авторизуватись в програмі.
   2.1. Ідентифікація користувачів відповідних ролей повинна здійснюватися за допомогою логіну та паролю, які вводяться на сторінці авторизації веб-додатку.
   2.2. При некоректному вводі з'являється повідомлення про помилку і прохання перевірити правильність введених логіну та паролю.
3. Користувач має можливість змінювати персональні дані
   3.1. Користувачу потрібно клікнути на власний аватар на верхній панелі, щоб відкрити меню користувача.
   3.2. У меню йому потрібно клікнути на пункт “Profile”.
   3.3. На сторінці “Profile” користувач може поміняти свій пароль, номер телефону, електронну адресу та власну фотографію.
4. Користувач має можливість переглядати каталог книг.
   4.1. Користувач може здійснювати фільтрування книг по таким параметрам: авторам, жанрам, рокам написання книги, мові написання книги, назві книги, ISBN книги, вибирати тільки електронні чи фізичні книги.
5. Користувач має можливість читати електронну книгу.

5.1. Користувач повинен вибрати електронну книгу серед каталогу книг.

5.2. На сторінці книги користувач повинен нажати на кнопку “Read”, після цього його перенаправить на вкладку, де він зможе читати книгу.

6. Користувач має можливість забронювати книгу.

   6.1. Користувач повинен вибрати книгу серед каталогу книг і перейти на сторінку книги.

   6.2. На сторінці книги йому потрібно нажати на кнопку “Order”, щоб замовити книгу. Після цього на електронну адресу прийе лист про те, що книга успішно замовлена.

   6.3 Бібліотекар відкладе книгу і користувач зможе її забрати, коли прийде до бібліотеки.

7. Користувач має можливість переглянути детальну інформацію про книгу.

   7.1. Користувач повинен вибрати книгу серед каталогу книг, навести на неї і нажати на “Details”.

8. Користувач має можливість переглядати всі свої замовлення.

   8.1. Користувач повинен нажати на свій аватар на верхній панелі, щоб відкрилося меню користувача.

   8.2. У меню користувач, користувач повинен нажати на “My orders” і після цього користувач може переглянути свої замовлення.

9. Бібліотекар має можливість додати книгу до каталогу, заповнивши всі необхідні поля форми для цього.

10. Бібліотекар має можливість віддати книгу користувачу.

    a. Забронювана книга.

       10.1. Бібліотекарю потрібно перейти на сторінку заброньованих книг.

10.2. Бібліотекар повинен відфільтрувати список заброньованих книг по користувачу, який прийшов забрати книгу.

10.3. Бібліотекар повинен нажати на поле книги у таблиці, щоб відкрити детальну інформацію про бронювання.

10.4. Бібліотекар повинен нажати на кнопку “Loan Book”, щоб записати у базу даних, що книгу взято і після цього віддати книгу користувачу.

b. Незабронювана книга.

10.1. Бібліотекар повинен знайти книгу, яку хоче взяти користувач, серед каталогу книг і перейти на сторінку книги.

10.2. На сторінці книги бібліотекар повинен нажати на кнопку “Loan”, щоб зберегти інформацію про взяття книги у БД, та віддати книгу користувачеві.

11. Бібліотекар має можливість переглядати статистику.

а. Бібліотекар може переглядати загальну статистику бібліотеки (кількість книжок, кількість взятих книжок за місяць та весь час, кількість користувачів, які заборгували книги).

б. Бібліотекар може переглядати статистику користувачів.

в. Бібліотекар може переглядати статистику книги.

12. Бібліотекар може переглядати список замовлених книг.

12.1. Бібліотекар може переглядати список всіх заброньованих книг або фільтрувати цей список за користувачем, датою та ін. Також бібліотекар може сортувати список по даті.

12.2. При натисканні на елемент списку відкриється детальна інформація про замовлення.

13. Бібліотекар може переглядати інформацію про взяття книг.

13.1. Бібліотекар може переглядати інформацію про всі взяття книг або фільтрувати цей список за користувачем, датою та ін. Також бібліотекар може сортувати список по даті.

13.2. При натисканні на елемент списку відкриється детальна і нформація про взяття книги.

14. Бібліотекар має можливість переглядати список всіх користувачів (крім бібліотекарів та адміністраторів).

14.1. Бібліотекар може переглядати список всіх користувачів чи здійснювати пошук за іменем, електронною поштою, номером телефону. Також бібліотекар може сортувати список по імені користувача, електронній адресі, номеру телефону, статусу користувача.

14.2. При натисканні на елемент списку відкриється детальна інформація про користувача.

14.3.1 Бібліотекар має можливість редагувати інформацію про користувача.

14.3.2. Бібліотекар має можливість видалити користувача.

15. Бібліотекар має можливість переглядати список всіх авторів у таблиці авторів на сторінці “Managing”. Також він може здійснювати пошук авторів по імені автора чи його країні.

а. При натисканні на іконку “смітник” користувач має можливість видалити автора.

б. При натисканні на іконку “олівець” відкриється вікно для редагування автора.

в. При натисканні кнопки “Add Author”, яка розміщена над таблицею, бібліотекар буде мати можливість додати автора після заповнення всіх необхідних даних форми.

16. Бібліотекар має можливість переглядати список всіх жанрів у таблиці жанрів на сторінці “Managing”. Також він може здійснювати пошук жанрів по імені жанру.

а. При натисканні на іконку “смітник” користувач має можливість видалити жанр.

б. При натисканні на іконку “олівець” відкриється вікно для редагування жнру.

в. При натисканні кнопки “Add Genre”, яка розміщена над таблицею, бібліотекар буде мати можливість додати жанр після заповнення всіх необхідних даних форми.

17. Бібліотекар має можливість зареєструвати користувача.

17.1. Бібліотекар повинен внести необхідну інформацію про користувача.

17.2. Користувач отримає електронний лист з даними для входу.

18. Бібліотекар має можливість перейти на детальну сторінку користувача.

18.1. Бібліотекар повинен перейти на список всіх користувачів.

18.2. Бібліотекар повинен вибрати користувача, натиснути на нього та нажати кнопку “More”, щоб перейти на детальну інформацію про користувача.

18.3. На сторінці користувача бібліотекар може переглянути інформацію про взяття книг користувача (має можливість їх фільтрувати та сортувати), його бронювання (має можливість їх фільтрувати та сортувати) та його статистику.

19. Бібліотекар має можливість переглядати розклад роботи бібліотекарів.

20. Адміністратор має можливість переглядати список всіх бібліотекарів.

20.1. Адміністратор може переглядати список всіх бібліотекарів чи здійснювати пошук за іменем, електронною поштою, номером телефону. Також адміністратор може сортувати список по імені бібліотекаря, електронній адресі, номеру телефону.

20.2. При натисканні на елемент списку відкриється детальна інформація про бібліотекаря.

20.3 Адміністратор має можливість редагувати інформацію про бібліотекаря.

20.4. Адміністратор має можливість видалити бібліотекаря.

21. Адміністратор має можливість добавляти бібліотекаря.

21.1. Адміністратор повинен внести необхідну інформацію про бібліотекаря.

21.2. Бібліотекар отримає електронний лист з даними для входу.

22. Адміністратор має можливість перейти на детальну сторінку бібліотекаря.

22.1. Адміністратор повинен перейти на список всіх бібліотекарів.

22.2. Адміністратор повинен вибрати бібліотекаря, натиснути на нього та нажати кнопку “More”, щоб перейти на детальну інформацію про бібліотекаря.

22.3. На сторінці бібліотекаря адміністратор може переглянути інформацію про книги, які віддав бібліотекар (має можливість їх фільтрувати та сортувати), його розклад та його статистику.

23. Адміністратор має можливість редагувати розклад роботи бібліотекарів.

24. Адміністратор має можливість видаляти розклад роботи бібліотекарів.

25. Адміністратор має можливість переглядати статистику бібліотекарів на сторінці “Statistic”.

26. Будь-який користувач має можливість вийти з акаунту.

3.2. Структура програми

Для централізованого зберігання інформації у програмі було використано документно-орієнтовану базу даних MongoDB. Логічна схема бази даних зображена на рис. 3.1.

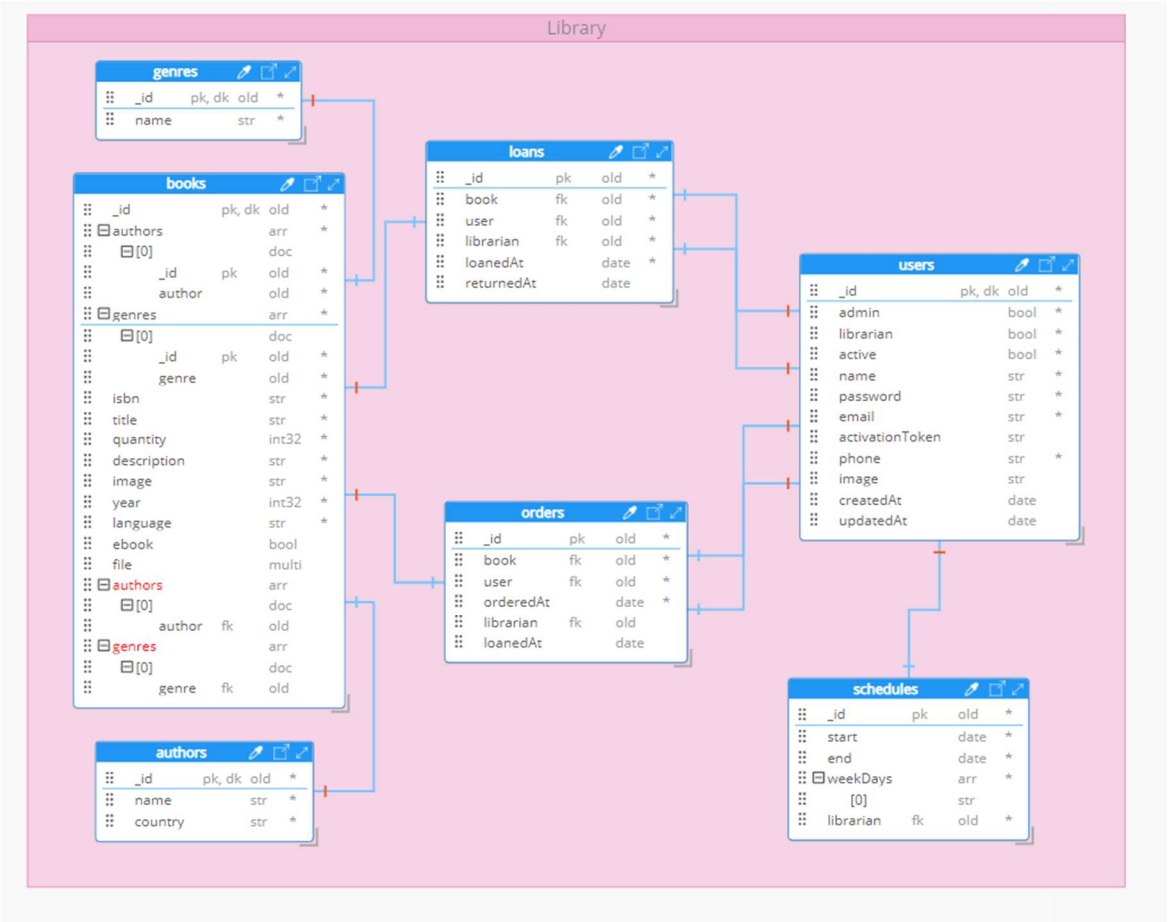


Рис. 3.1. Схема бази даних

База даних складається з таких таблиць (рис. 3.2 – рис. 3.8).

1. Users – зберігає інформацію про всіх користувачів.
2. Schedules – зберігає інформацію про розклад бібліотекарів.
3. Books – зберігає інформацію про всі книги.
4. Authors – зберігає інформацію про всіх авторів.
5. Genres - зберігає інформацію про всі жанри.
6. Loans - зберігає інформацію про всі взяття книг.
7. Orders - зберігає інформацію про всі бронювання.

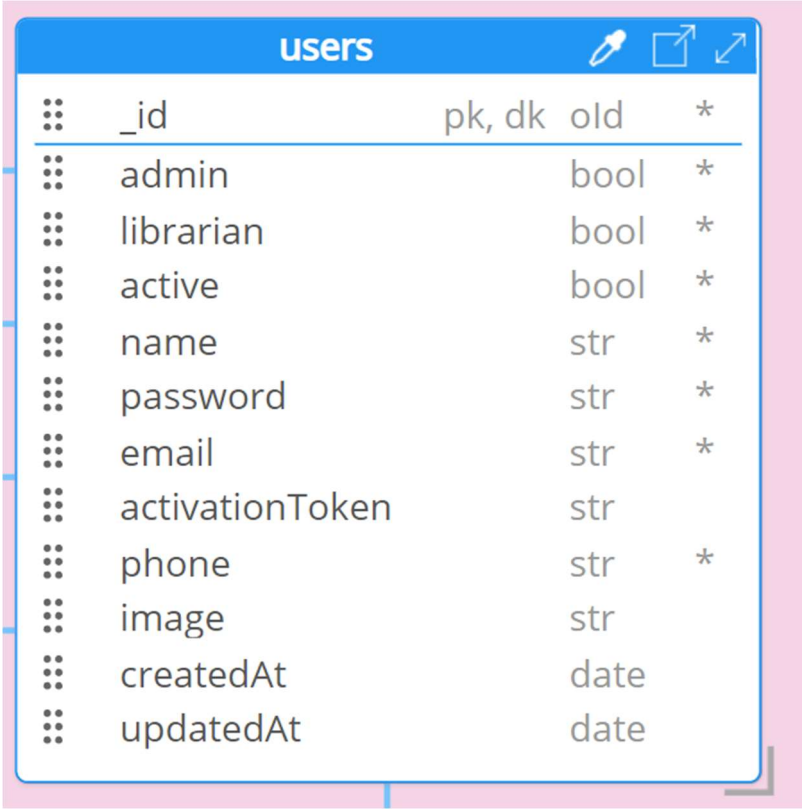


Рис. 3.2. Сутність Users

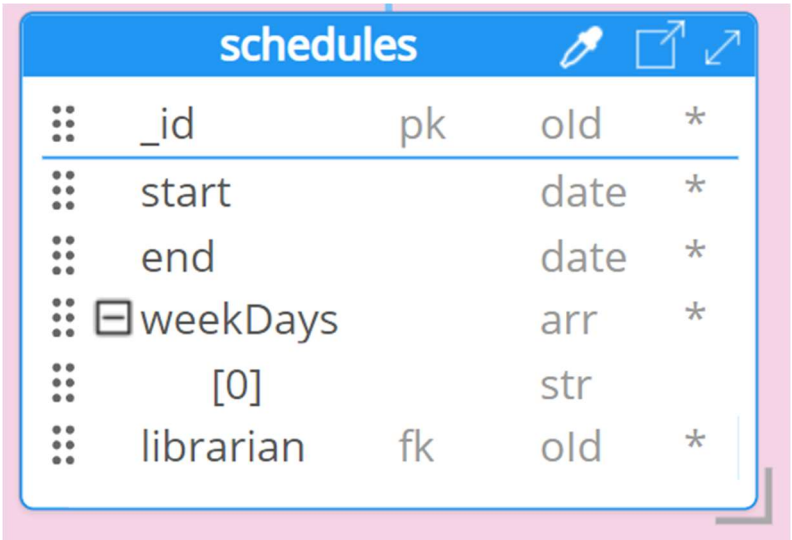


Рис. 3.3. Сутність Schedules

Рис. 3.4. Сутність Books

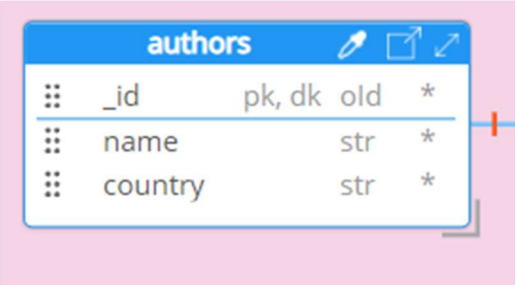


Рис. 3.5. Сутність Authors

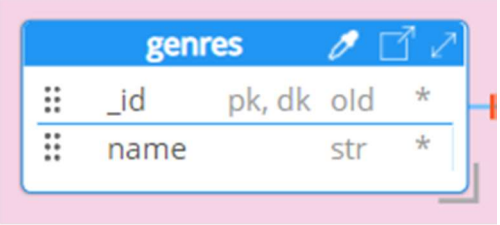


Рис. 3.6. Сутність Genres

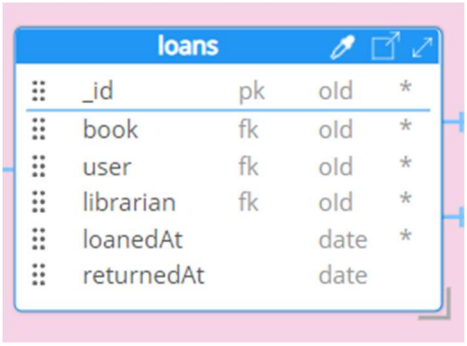


Рис. 3.7. Сутність Loans

Рис. 3.8. Сутність Orders

Узагальнена діаграма прецедентів програми зображена на рисунку 3.9.

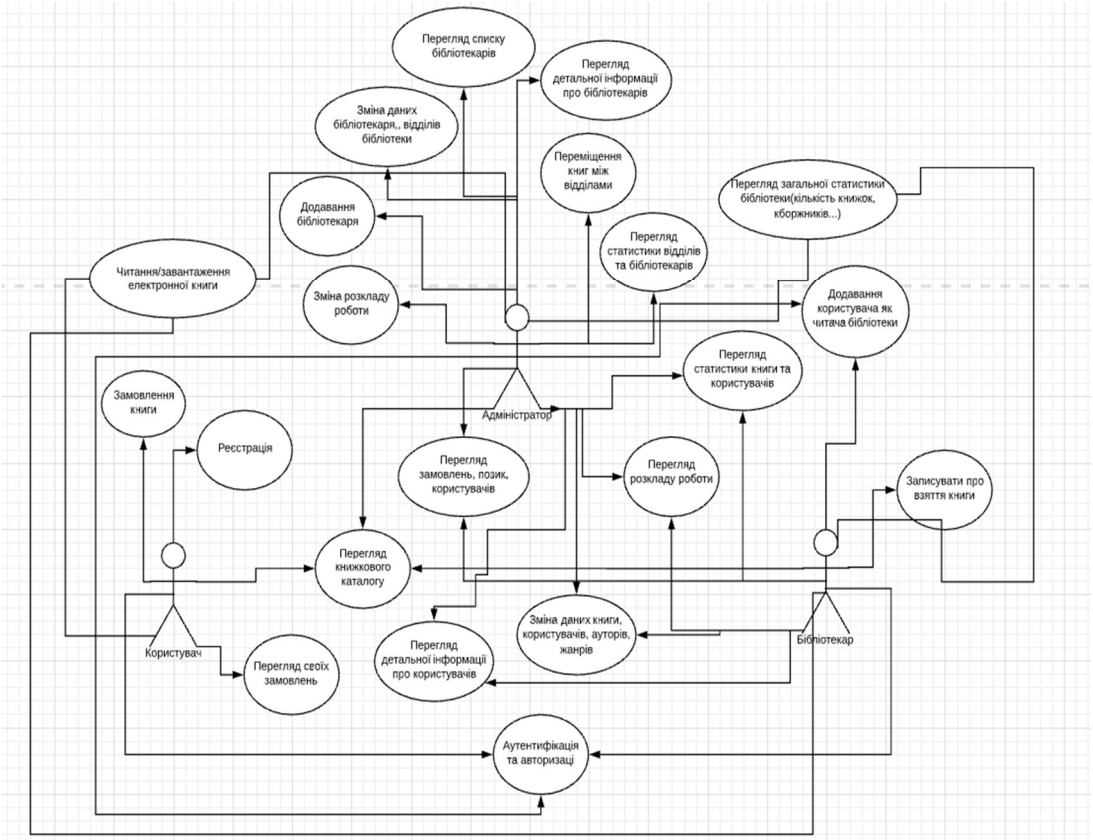


Рис. 3.9. Діаграма прецедентів

3.3. Опис функціоналу та інтерфейсу користувача створеного веб-додатку

Потрібно перейти за посиланням Library, щоб відкрити веб-додаток. При відкритті з'явиться домашня сторінка (рис. 3.10), на якій розміщена основна інформація про бібліотеку.

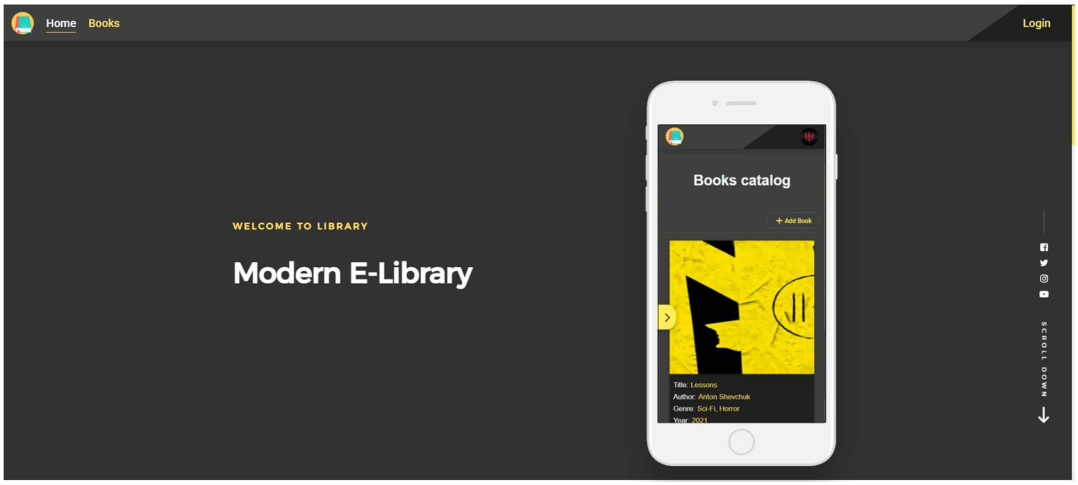


Рис. 3.10. Домашня сторінка

Якщо користувач хоче ознайомитися тільки з каталогом книг, то він може перейти на сторінку «Books» (рис. 3.11).

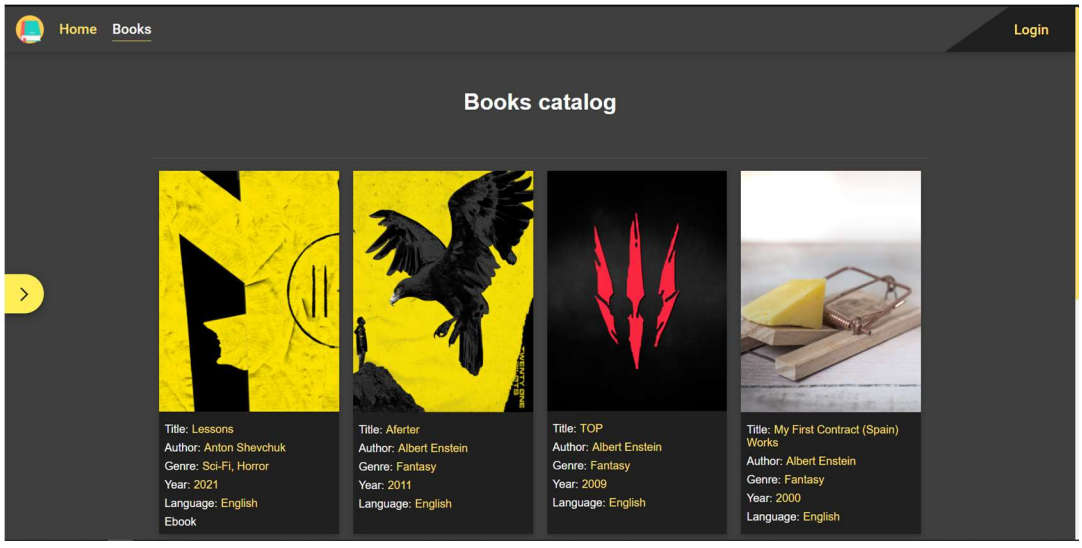


Рис. 3.11. Каталог книг

Користувач може скористатися фільтром для пошуку необхідних йому книг (рис. 3.12).

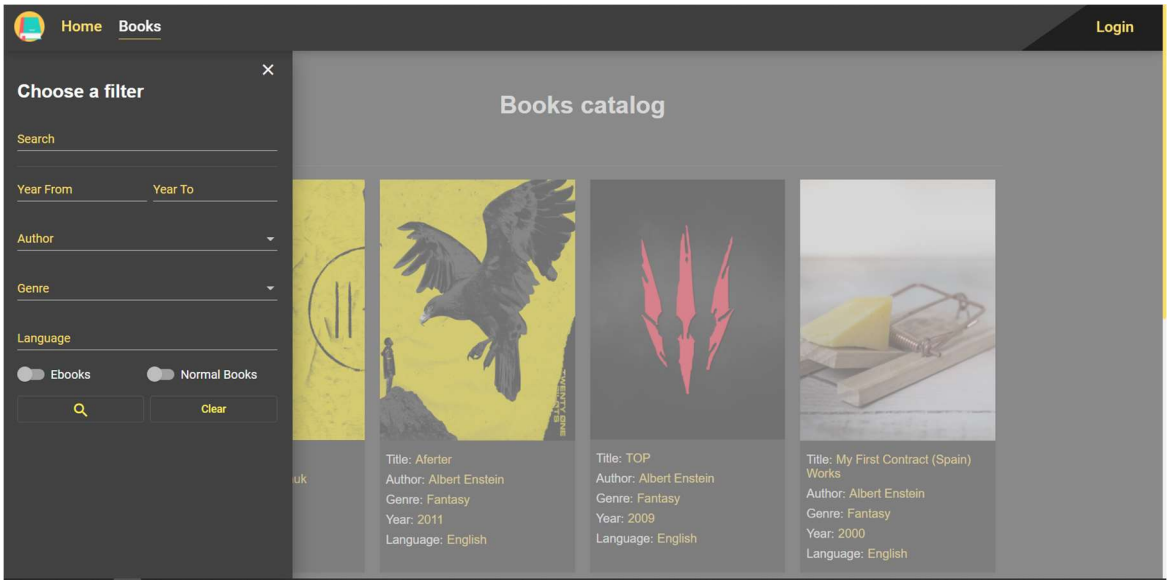


Рис. 3.12. Фільтр для пошуку книг

Користувач може здійснювати пошук здійснювати за декількома фільтрами (рис. 3.13).

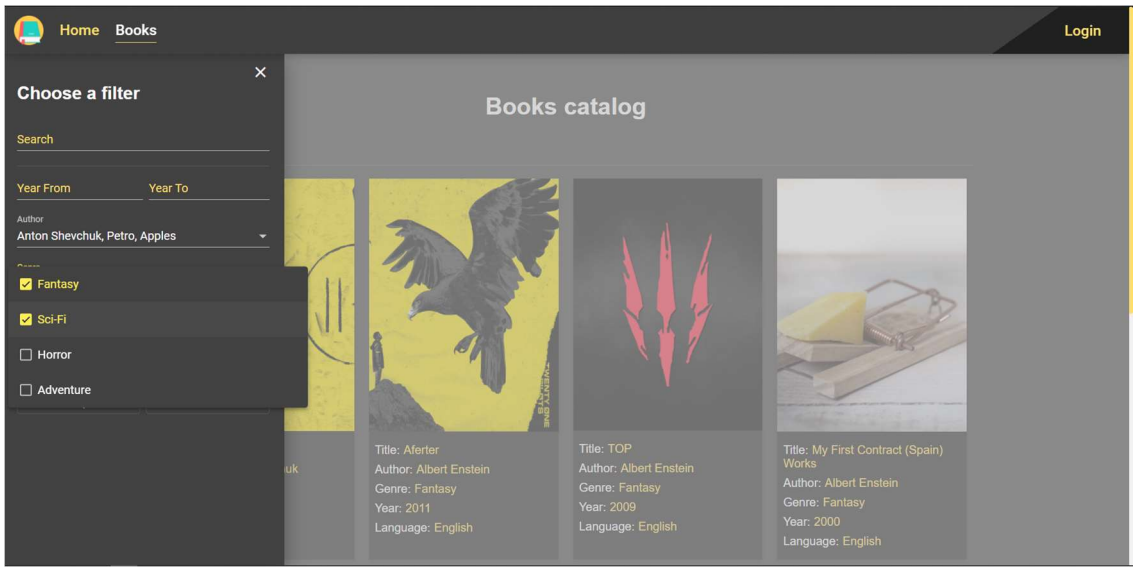


Рис. 3.13. Фільтр для пошуку книг з вибраними полями

Також користувач може переглянути детальну інформацію про книгу. Для цього йому потрібно навести на потрібну книгу, тоді з'явиться кнопка «Details» (рис. 3.14), натиснути на неї і користувача перенаправить на сторінку книги (рис. 3.15).

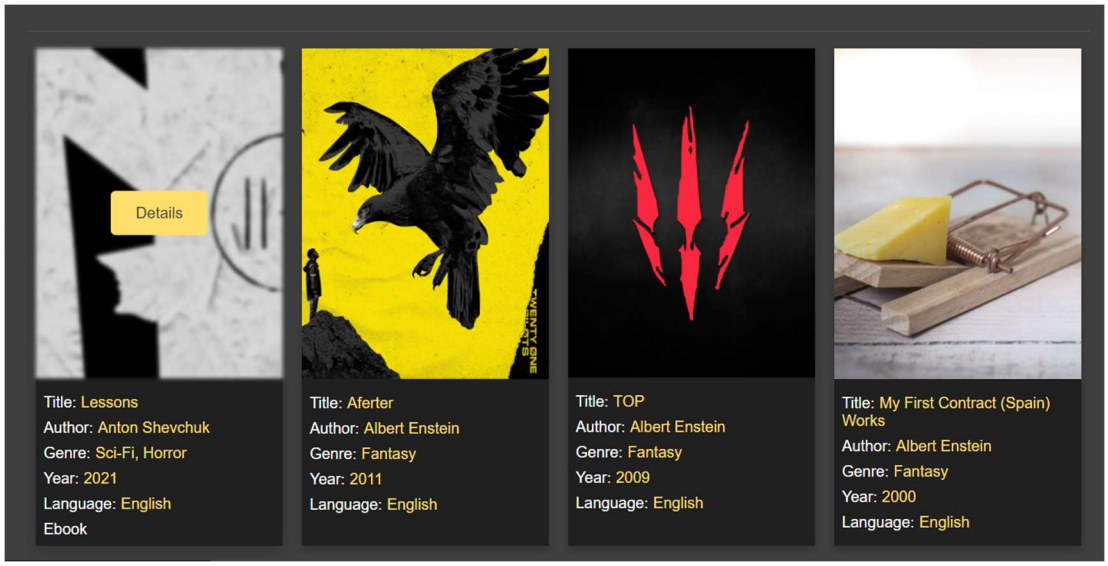


Рис. 3.14. Книга при наведенні

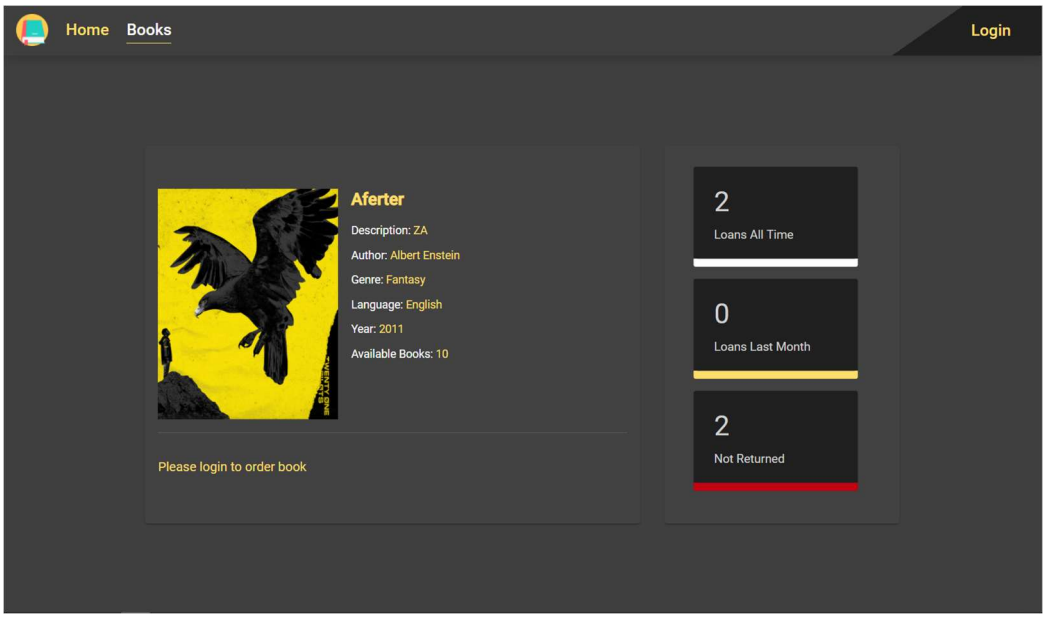


Рис. 3.15. Сторінка книги

На рисунку 3.15 користувач не авторизований і йому виводить повідомлення про те, щоб замовити книгу йому спочатку потрібно авторизуватися. На сторінці він бачить детальну інформацію про книгу, а також статистику за останній місяць і весь час.

Для того, щоб авторизуватися потрібно натиснути на кнопку «Login» на верхній панелі (рис. 3.16), після цього користувач перейде на сторінку авторизації (рис. 3.17).

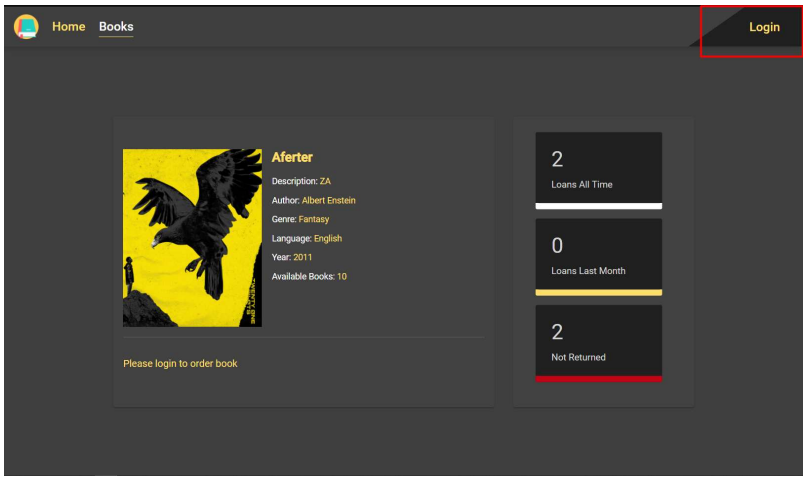


Рис. 3.16. Кнопка «Login»

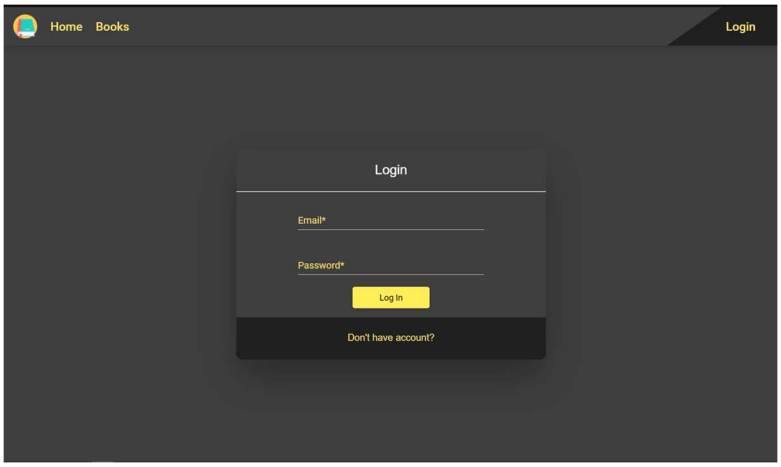


Рис. 3.17. Сторінка авторизації

На сторінці авторизації користувач має дві опції: а) авторизуватися; б) натиснути на посилання «Don’t have account?» і перейти на сторінку реєстрації (рис. 3.18).

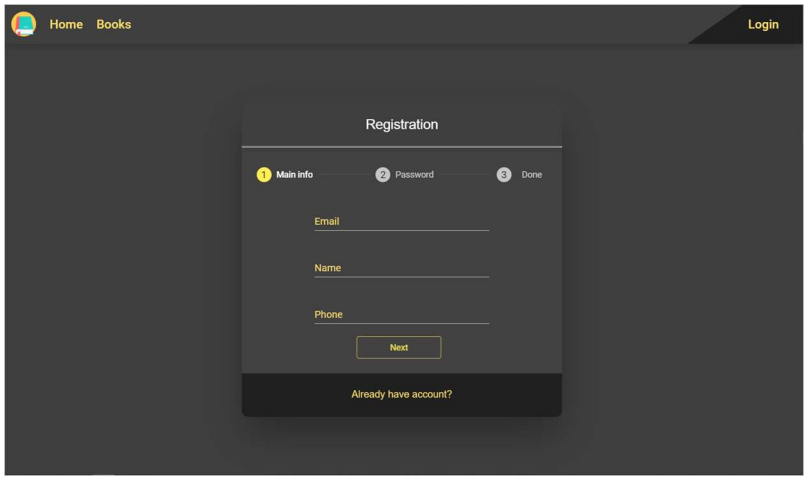
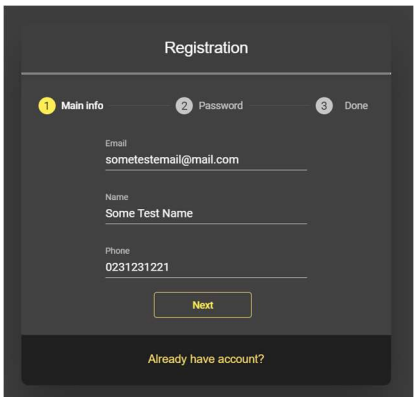


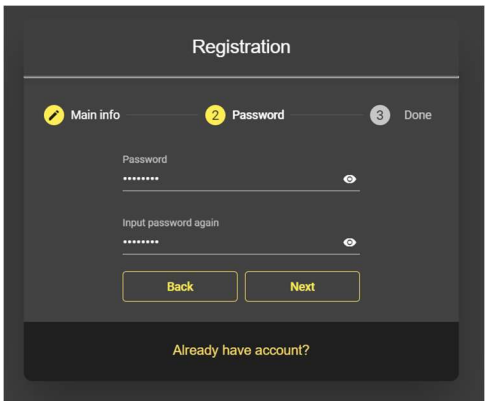
Рис. 3.18. Сторінка реєстрації

Для того, щоб зареєструватися потрібно пройти всі етапи реєстрації (рис. 3.19). Також користувач може повернутися на сторінку авторизації, натиснувши на посилання «Already have account?»

а)



б)



в)

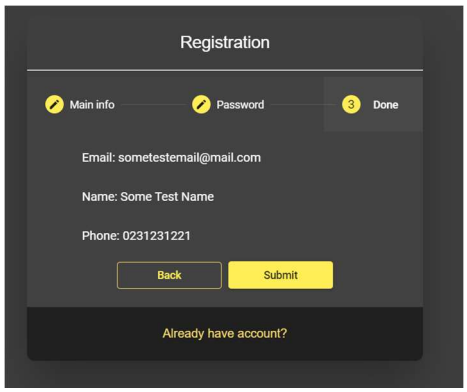


Рис. 3.19. Етапи реєстрації: а) Етап «Головна інформація»; б) Етап «Пароль»; в) Етап «Завершення»

Для завершення реєстрації користувачу потрібно натиснути на кнопку «Submit». Після завершення реєстрації користувача буде додано у БД, але він буде неактивний і не зможе увійти на сайт. Для входу користувачу потрібно підтвердити свою пошту, прийде лист на електронну пошту (рис. 3.20), і тільки після того його акаунт буде активним

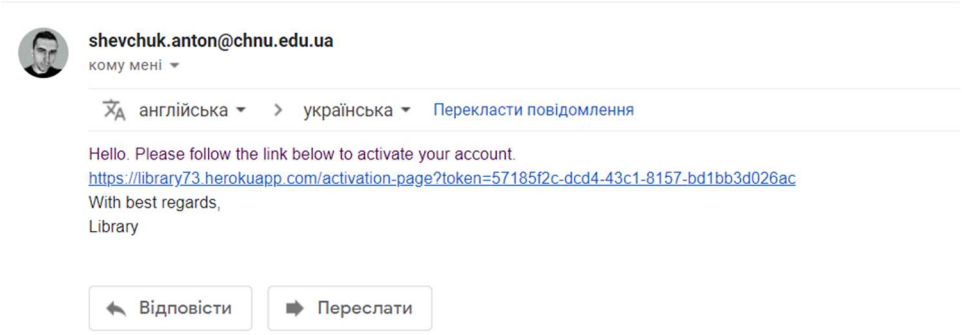
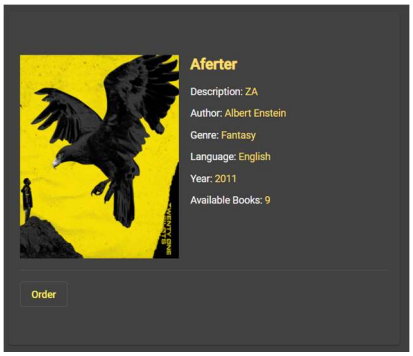


Рис. 3.20. Лист про активацію облікового запису

При кліку на посилання користувач переходить на сторінку активації і якщо все успішно, то він побачить повідомлення про те, що акаунт активований успішно.

Після того як користувач авторизувався у нього з'являється більше можливостей. Тепер користувач має можливість замовити книгу, або почати читати її, якщо це електронна книга (рис. 3.21). Якщо користувач замовив книгу, то йому на електронну адресу прийде лист про те, що книга успішно замовлена.

а) 

б) 

Рис. 3.21. Можливості користувача на сторінці книги: а) Замовлення книги; б) Читання книги

Для того, щоб замовити книгу потрібно натиснути на кнопку «order» і підтвердити про замовлення книги. Для того, щоб почати читати книгу потрібно натиснути на кнопку «Read» і після цього користувача направить на сторінку, де він зможе почитати або завантажити книгу.

Для перегляду власної інформації користувачу потрібно перейти на власну сторінку. Для цього потрібно натиснути на аватарку на верхній панелі, після чого відкриється меню. Серед опцій меню буде «Profile» (рис. 3.22). При натисканні на «Profile» користувача перенаправить на його сторінку (рис. 3.23).

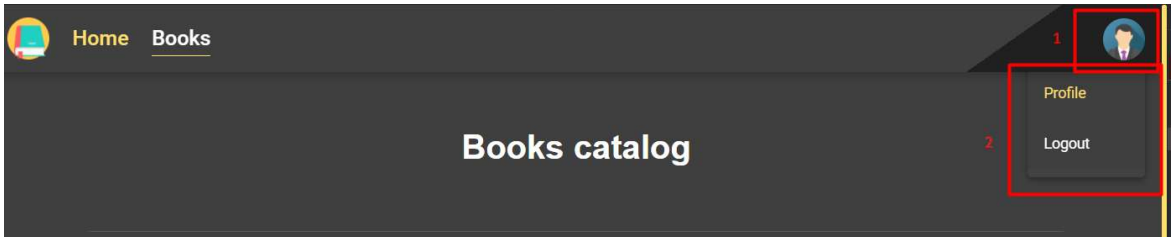


Рис. 3.22. Меню

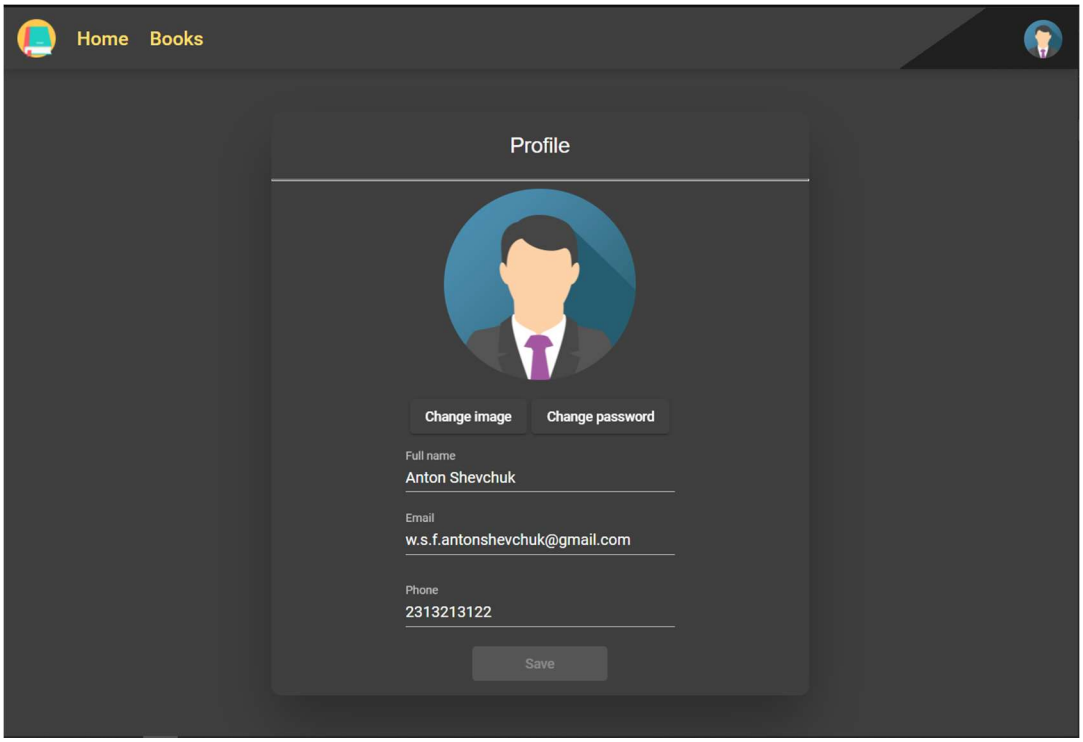


Рис. 3.23. Профіль користувача

На даній сторінці користувач може змінювати основну інформацію, а також може змінити зображення (рис. 3.24) та пароль (рис. 3.25). Користувач сам обирає як обрізати зображення.

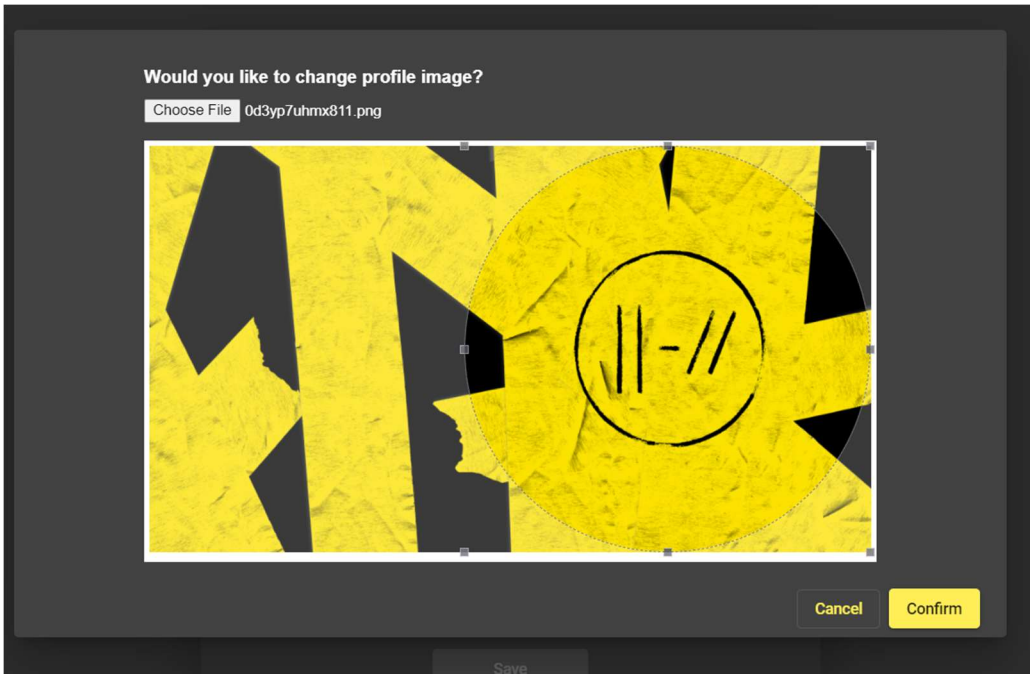


Рис. 3.24. Діалогове вікно для зміни аватарки користувача

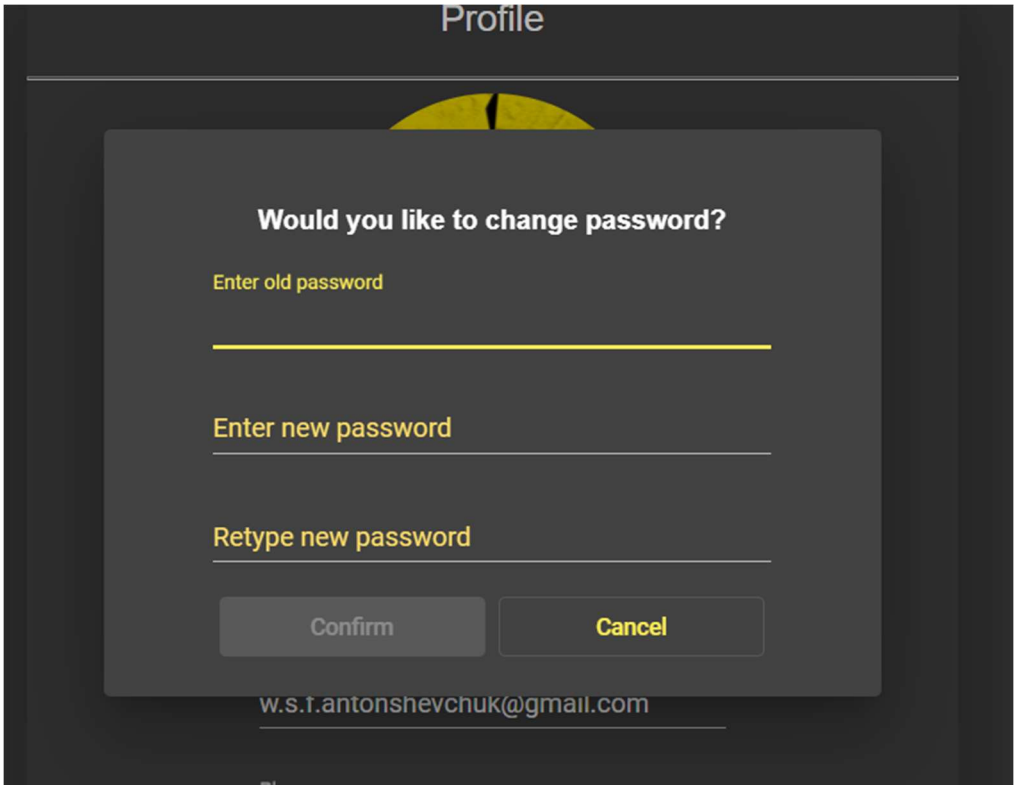


Рис. 3.25. Діалогове вікно для зміни пароля користувача

Бібліотекарю доступно більше можливостей. Перш за все, на на сторінці каталогу книг з’являється кнопка «Add Book» (рис. 3.26).

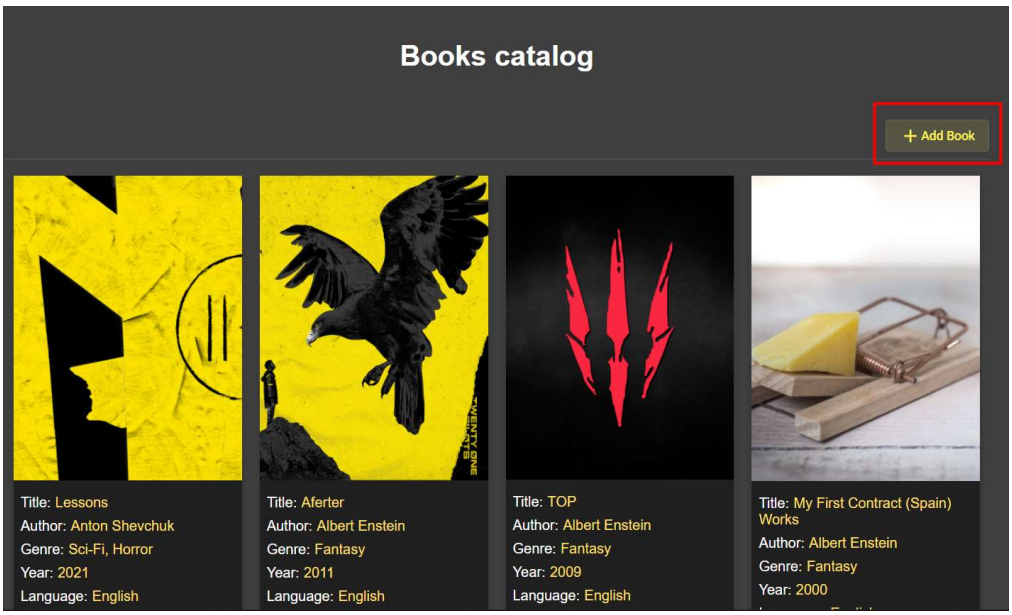


Рис. 3.26. Кнопка для додавання книги

Після натискання на кнопку відкриється діалогове вікно з формою для додавання книги (рис. 3.27).

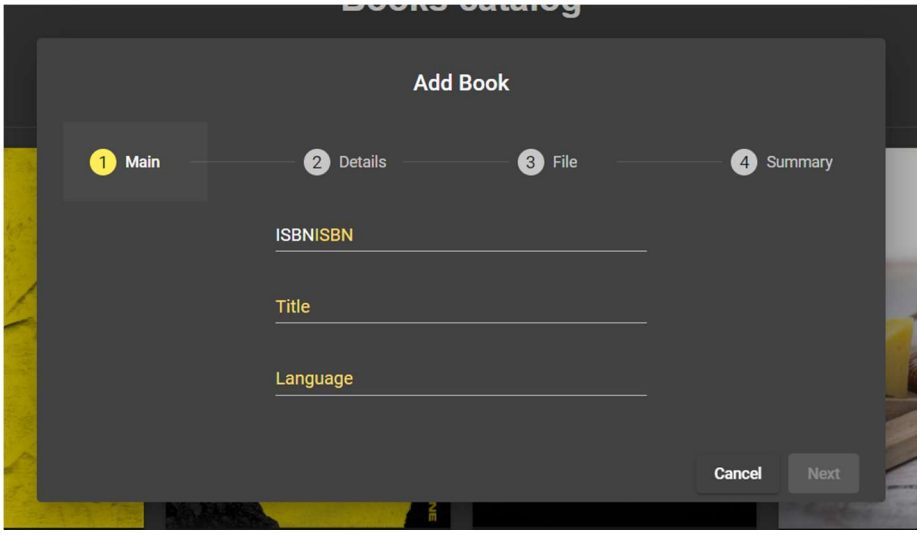
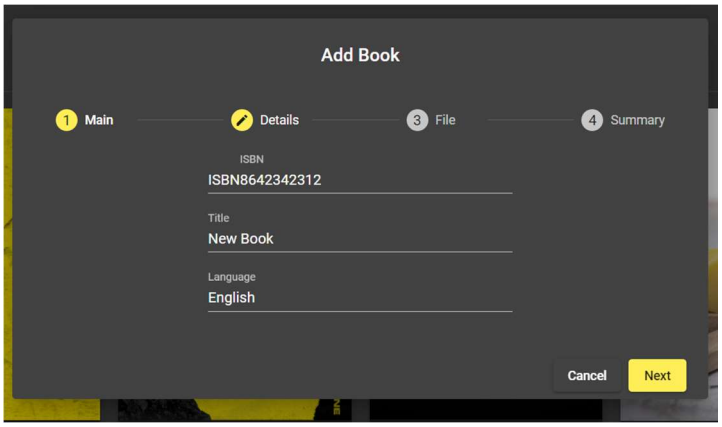


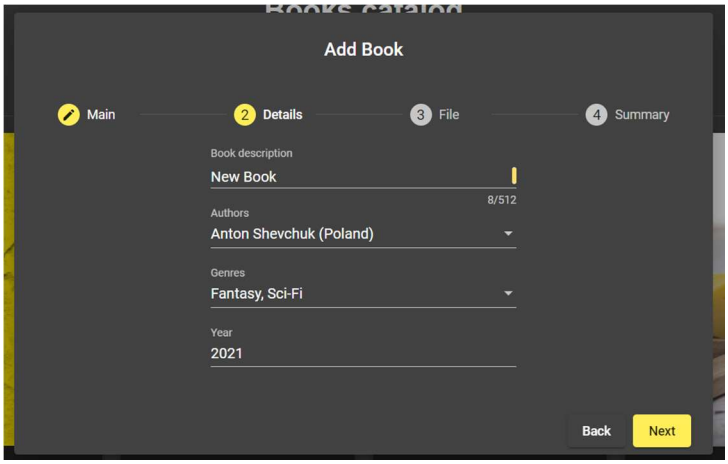
Рис. 3.27. Форма для додавання книги

Для того, щоб додати книгу потрібно заповнити всі поля, пройти всі кроки (рис. 3.28).

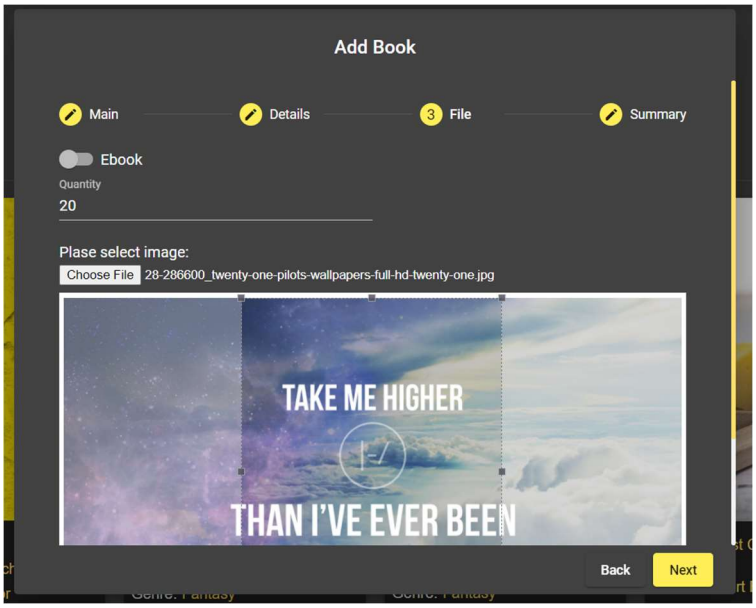
а)



б)



в)



г)

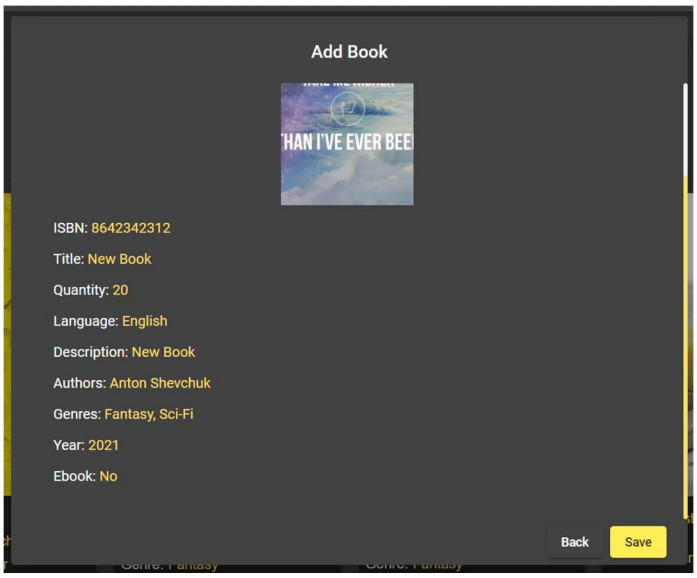


Рис. 3.28. Кроки створення книги: а) Головний; б) Деталі; в) Файл; г) Підсумок

Після проходження всіх кроків потрібно натиснути кнопку «Save» і книгу буде додано.

На сторінці книги у бібліотекаря інші можливості, ніж у користувача (рис. 3.29).

Рис. 3.29. Можливості бібліотекаря

Бібліотекар може дати книгу користувачеві (натиснути «Loan»), редагувати книгу (натиснути «Edit»), видалити книгу (натиснути «Delete»). Якщо бібліотекар натиснув кнопку «Loan», то перед ним з'являється діалогове вікно (рис. 3.30).

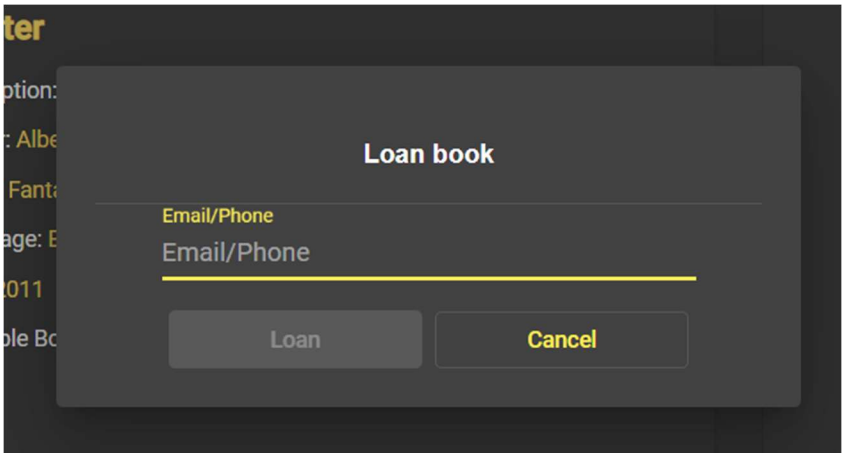


Рис. 3.30. Діалогове вікно для видання книги

Щоб дати книгу читачу, бібліотекарю потрібно ввести дані користувача: електронну адресу чи телефон. Після вводу даних натиснути «Loan».

Якщо бібліотекар натискає кнопку «Edit», то перед ним з'явиться діалогове вікно (рис. 3.31) із заповненою формою (форма заповнена даними книги).

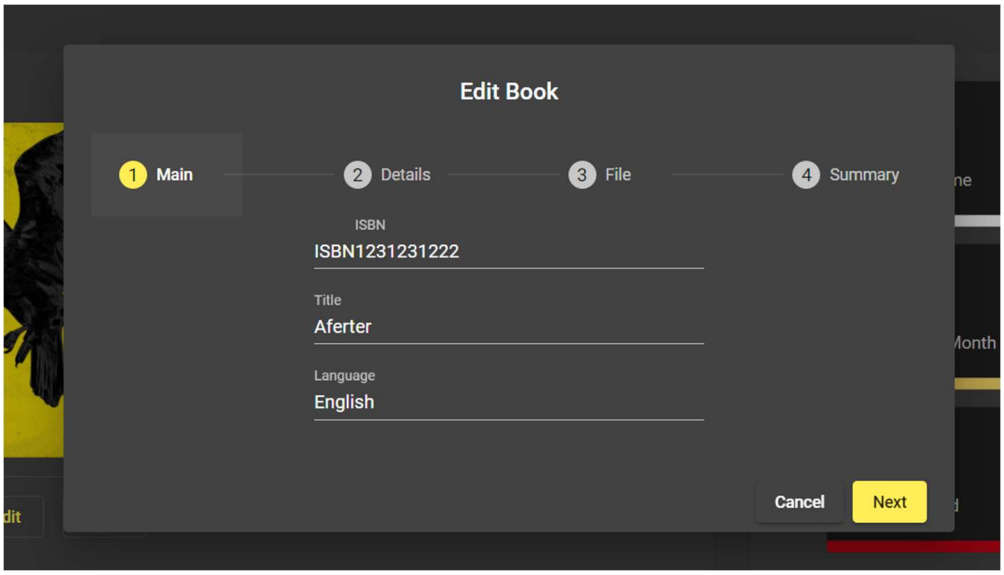
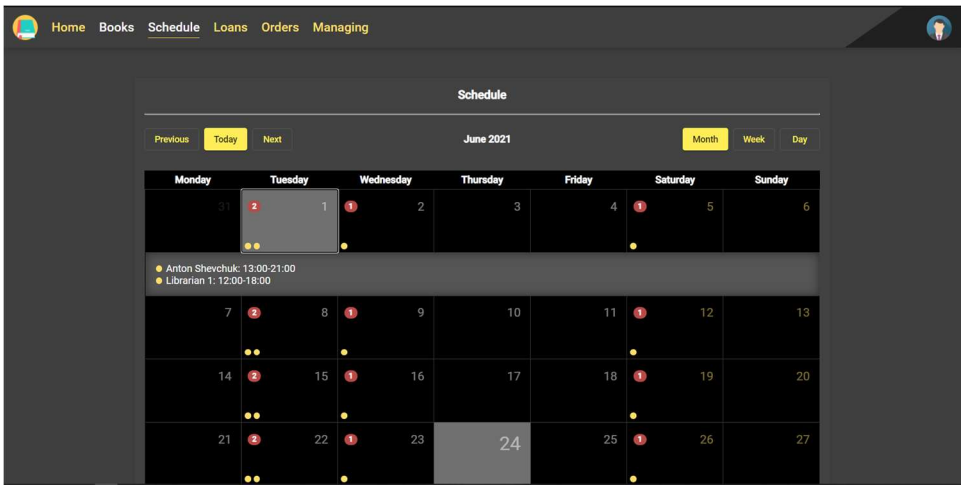


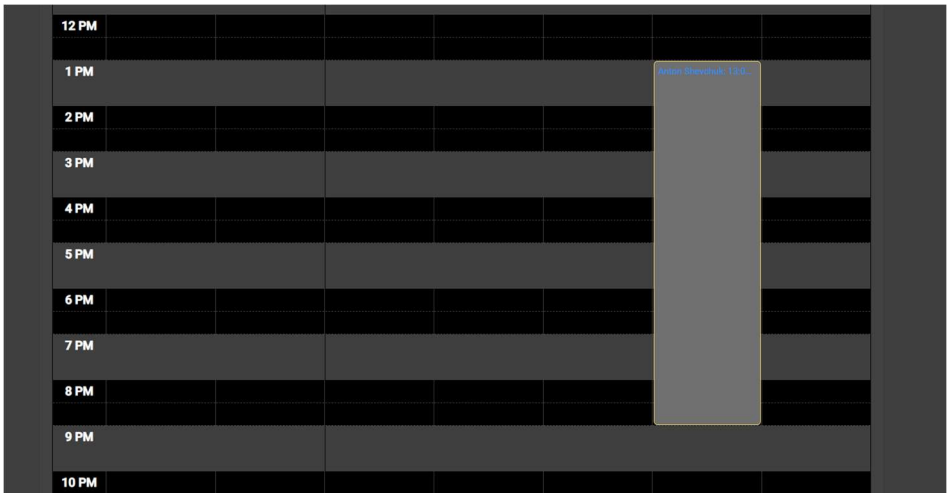
Рис. 3.31. Форма редагування із заповненими даними книги

Бібліотекар може переглядати розклад роботи (місяць, тиждень, день) свій та інших бібліотекарів (рис. 3.32).

а)



б)



в) 

Рис. 3.32. Розклад роботи бібліотекарів: а) Вид «Місяць»; б) Вид «Тиждень»; в) Вид «День»

Також бібліотекар може переглядати інформацію про всі взяття книг (рис. 3.33) і він може сортувати інформацію по даті виданню книги, а також може фільтрувати цю інформацію за різними фільтрами (Рис. 3.34).

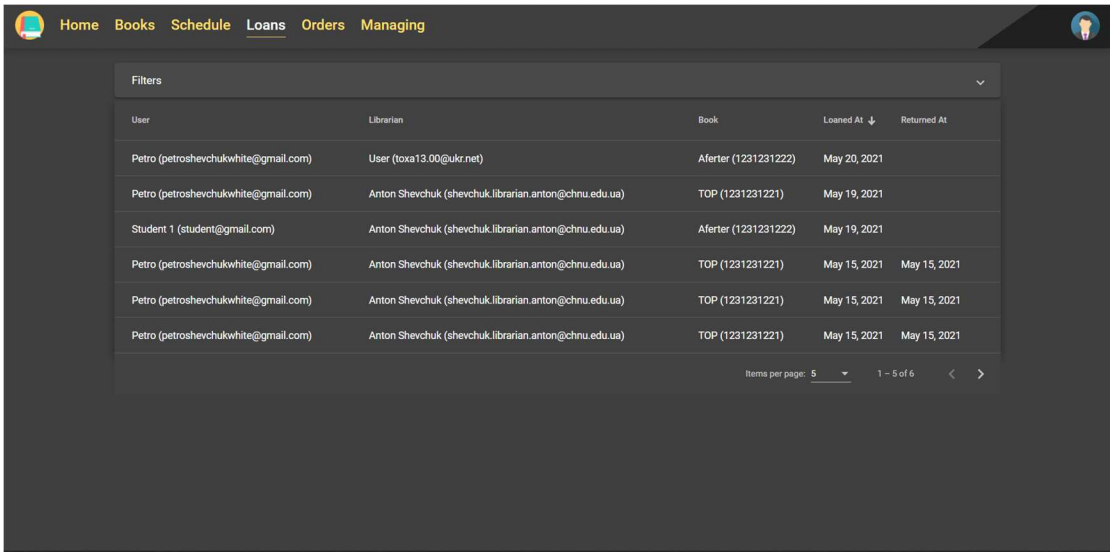


Рис. 3.33. Інформація про взяття книг читачами

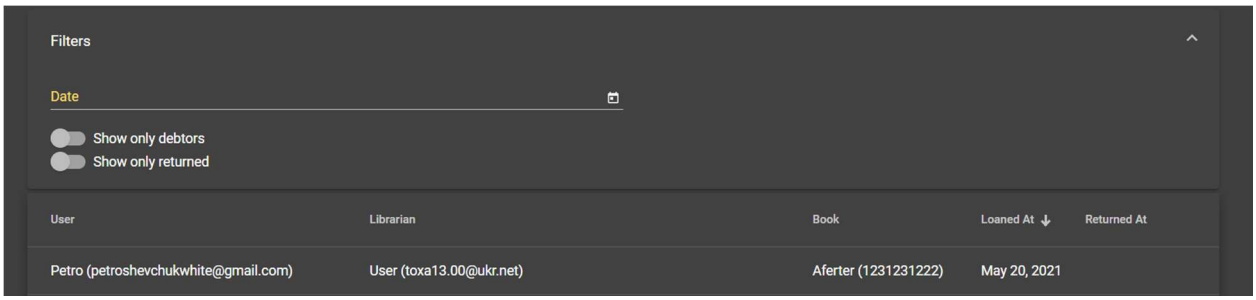


Рис. 3.34. Фільтр для фільтрування інформації про взяття книг

При кліку на елемент таблиці, бібліотекар буде бачити більш детальну інформацію про взяття книги (рис. 3.35).

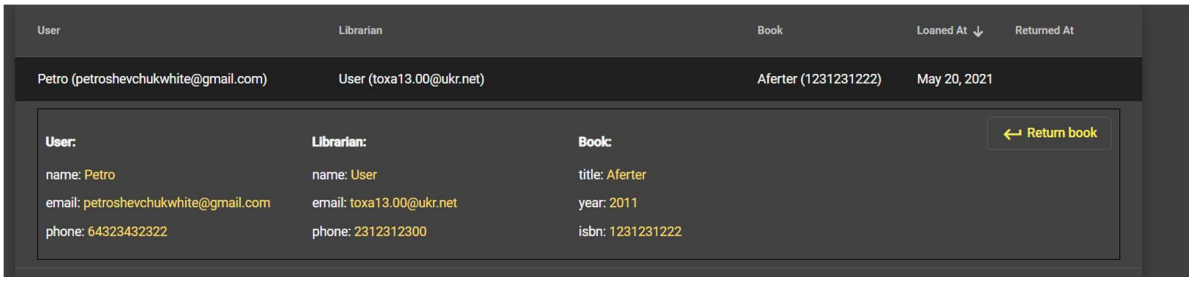


Рис. 3.35. Детальна інформація про взяття книги

Бібліотекар може повернути книгу, якщо вона ще не повернута, натиснувши на кнопку «Return book».

Бібліотека може переглядати список всіх заброньованих книг. Також він може сортувати по даті, фільтрувати та при кліку дивитися на детальну інформацію бронювання (рис. 3.36).

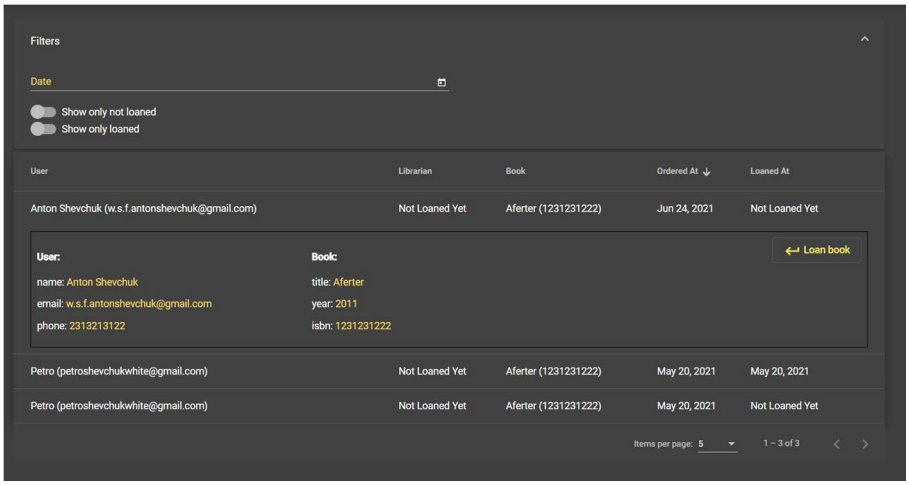


Рис. 3.36. Список заброньованих книг з відкритим фільтром та детальною інформацією про бронювання

Також бібліотекар одразу може віддати книгу користувачеві на цій сторінці, натиснувши на кнопку «Loan book».

Бібліотека має можливість переглядати загальну статистику по бібліотеці, а також статистику по користувачу чи книзі (рис. 3.37, рис. 3.38).

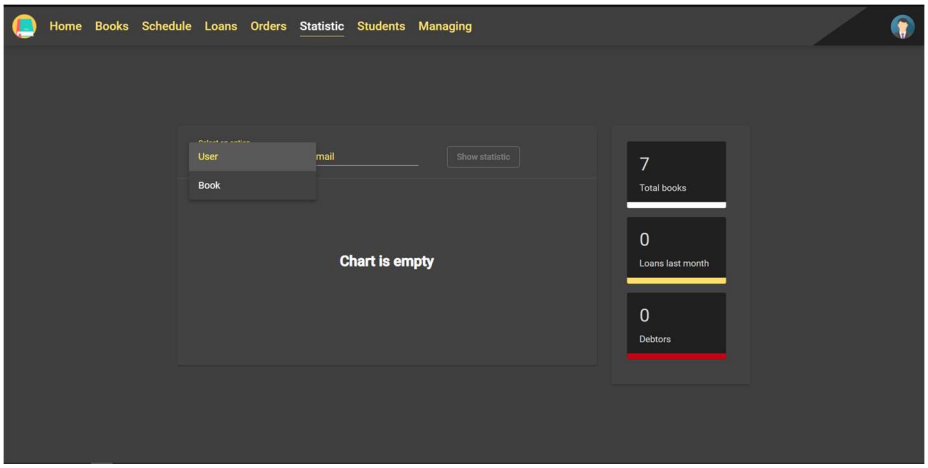


Рис. 3.37. Сторінка статистики

Рис. 3.38. Статистика по користувачу

Бібліотекар може переглядати список всіх користувачів (рис. 3.39). Бібліотекар може сортувати цей список по всім полям таблиці та виконувати пошук по всім полям таблиці.

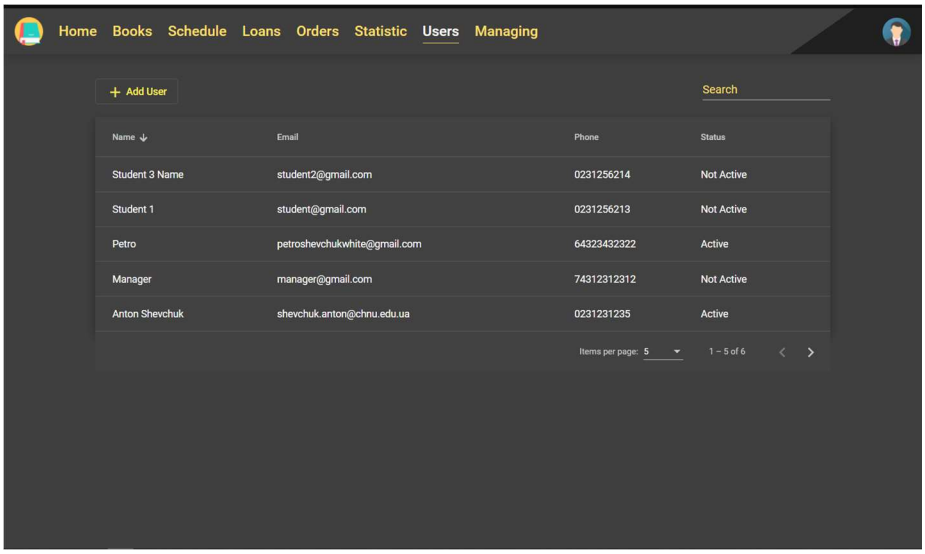


Рис. 3.39. Список всіх користувачів

Також бібліотекар може додавати користувача. Для цього потрібно натиснути на кнопку «Add user» і перед ним появиться діалого вікно для вводу даних студента (рис. 3.40).

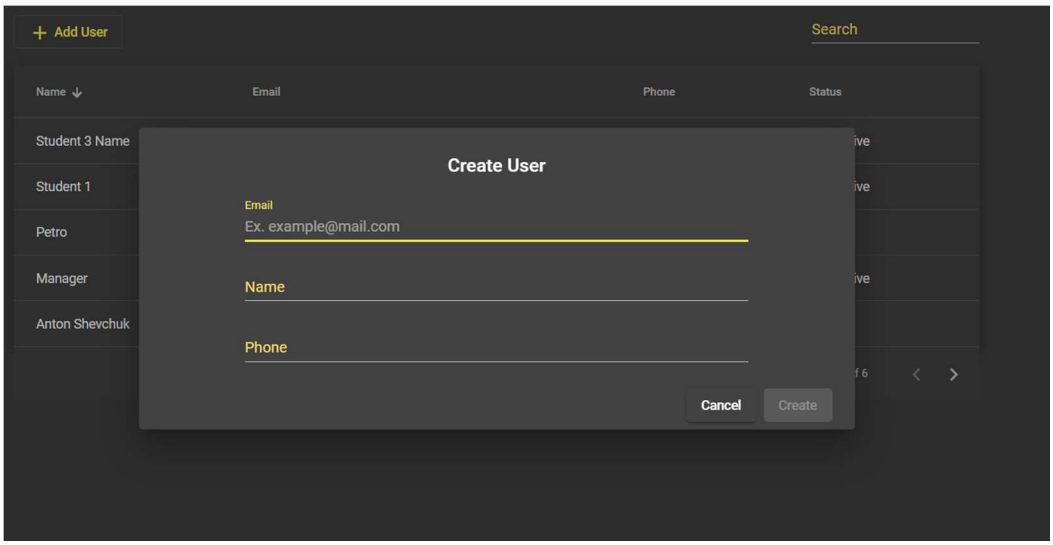


Рис. 3.40. Форма для додавання читача

Після заповнення форми користувачу на електронну адресу прийдуть дані для входу.

Також він може при кліку на користувача побачити більш детальну інформацію про нього (рис. 3.41).

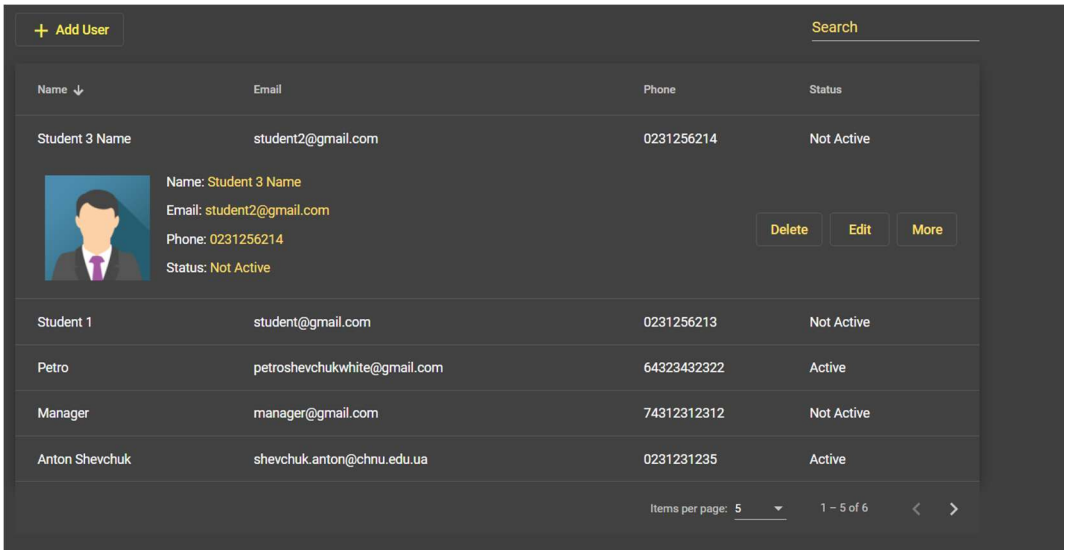


Рис. 3.41. Більш детальна інформація про користувача

Після відкриття більш детальної інформації про користувача бібліотекар може змінити дані користувача (натиснути «Edit»), видалити користувача

(натиснути «Delete») або перейти на сторінку користувача (натиснути «») (рис. 3.42).

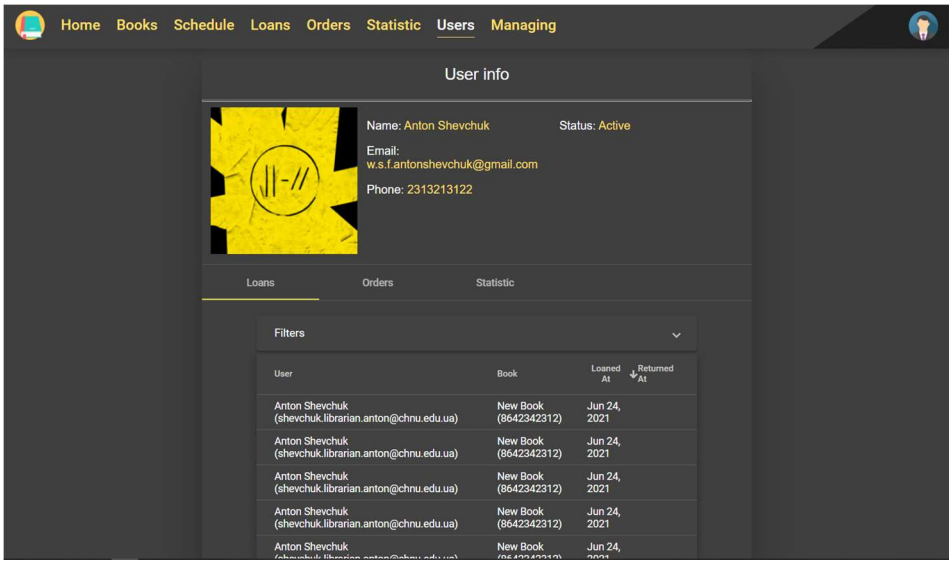
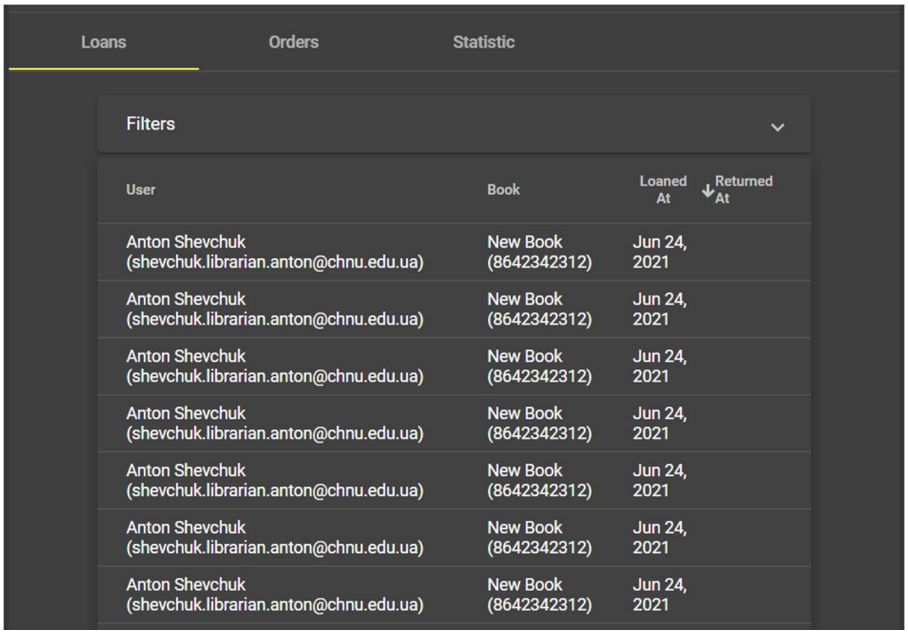


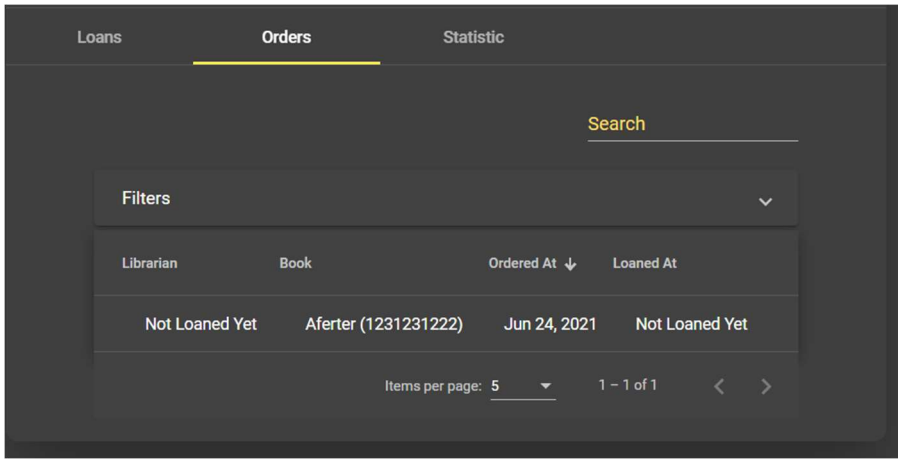
Рис. 3.42. Сторінка користувача

На сторінці користувача, крім даних користувача, бібліотекар може побачити інформацію про взяття книг користувачем, його замовлення та статистику (рис. 3.43).

а)



б)



в)

Рис. 3.43. Додаткова інформація про користувача

У бібліотекаря є сторінка управління (рис. 3.44), де він може переглядати список всіх авторів, жанрів, міняти, додавати, видаляти, сортувати, шукати ці сутності.

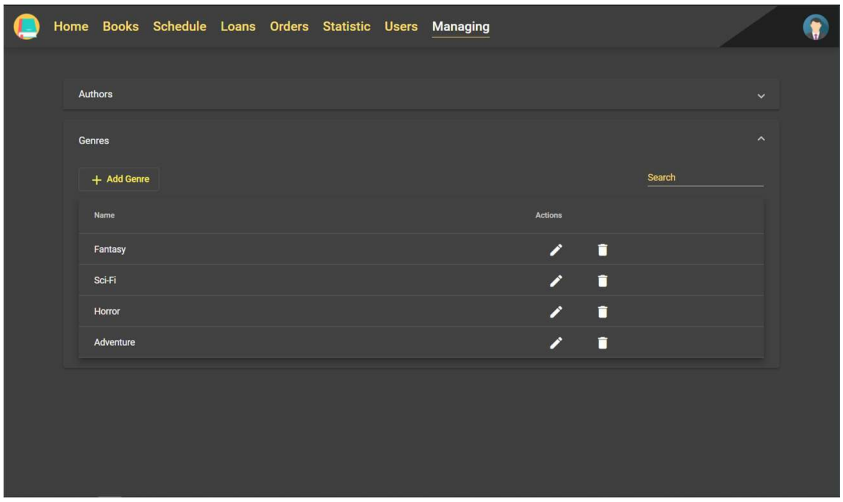


Рис. 3.44. Сторінка управління

На відмінно від бібліотекаря адміністратор має можливість додавати розклад роботи бібліотекарям. Для цього йому потрібно натиснути на кнопку «Add Schedule» на сторінці розкладу (рис. 3.45).

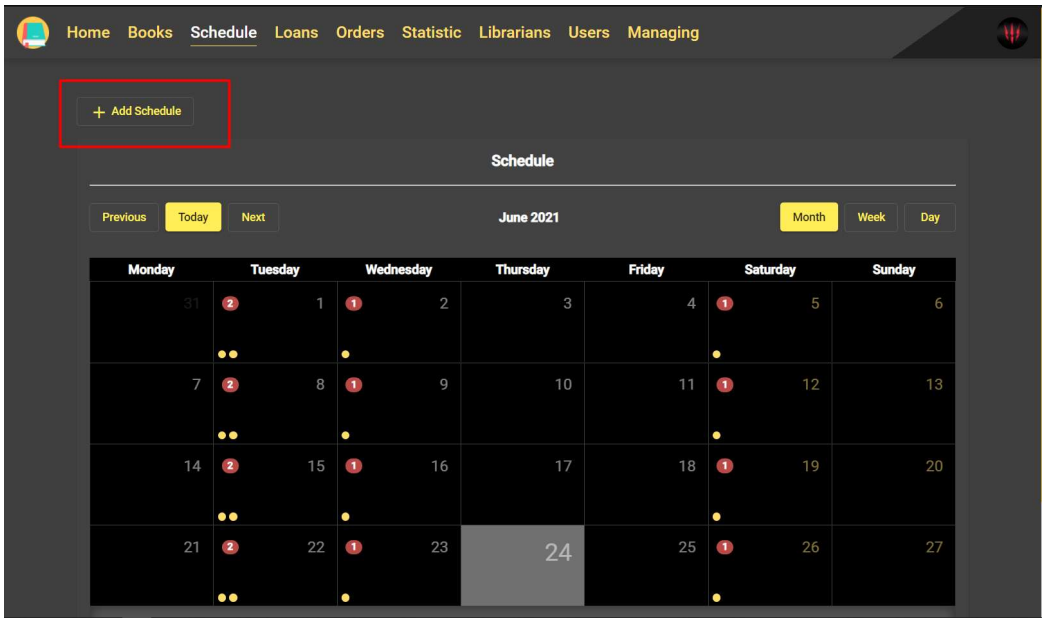


Рис. 3.45. Сторінка розкладу з кнопкою для додавання розкладу

Після натискання кнопки з’являється форма для вводу розкладу (рис. 3.46).

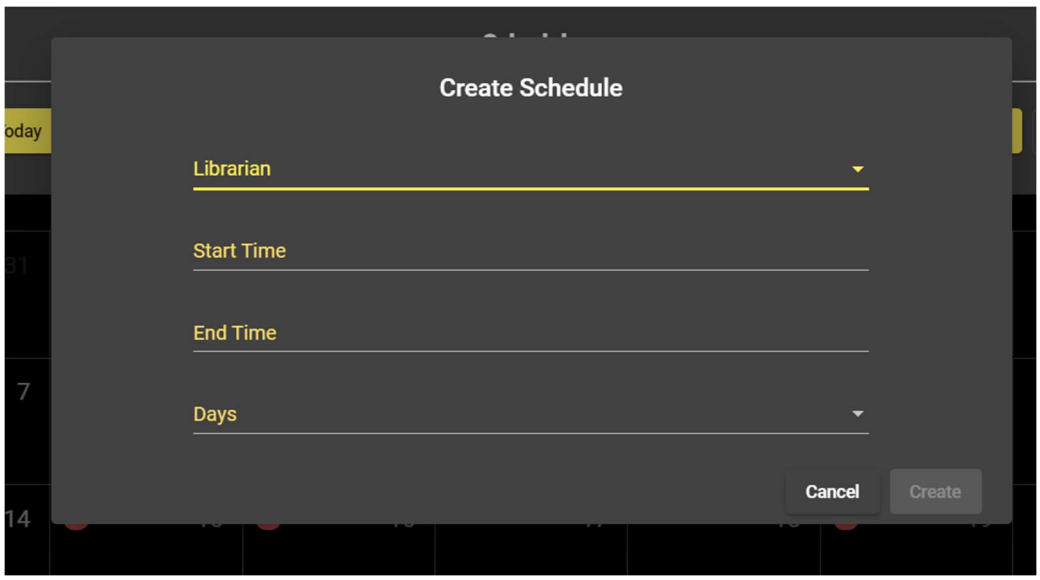


Рис. 3.46. Форма для вводу розкладу

Також на сторінці статистики адміністратор може переглядати статистику бібліотекаря (рис. 3.47).

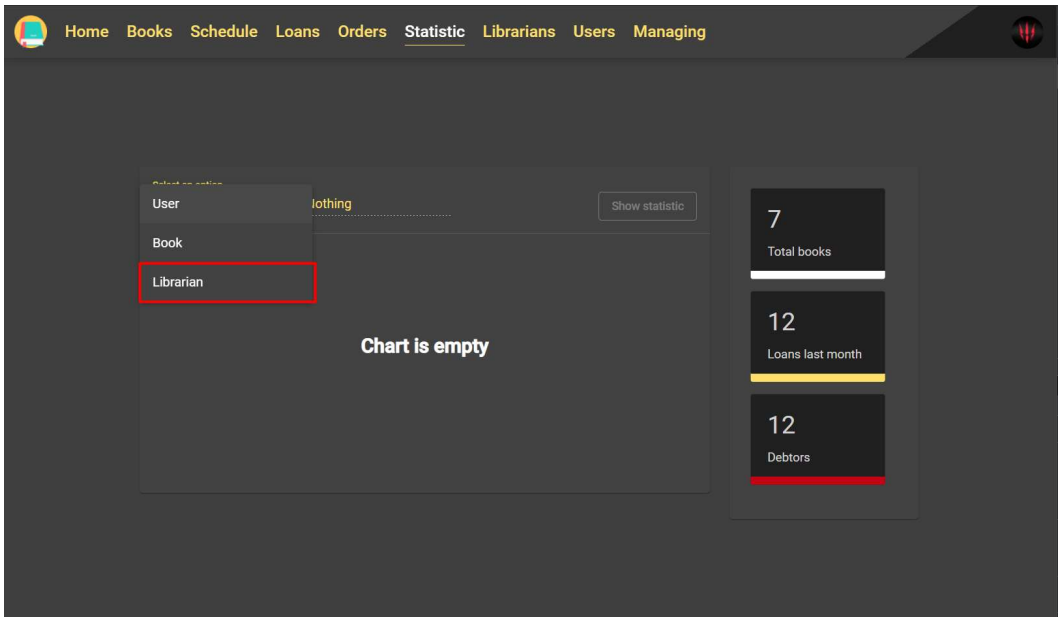


Рис. 3.47. Сторінка статистики

Адміністратор має можливість переглядати всіх бібліотекарів (рис. 3.48). Ця сторінка з такими ж функціями як сторінка користувачів.

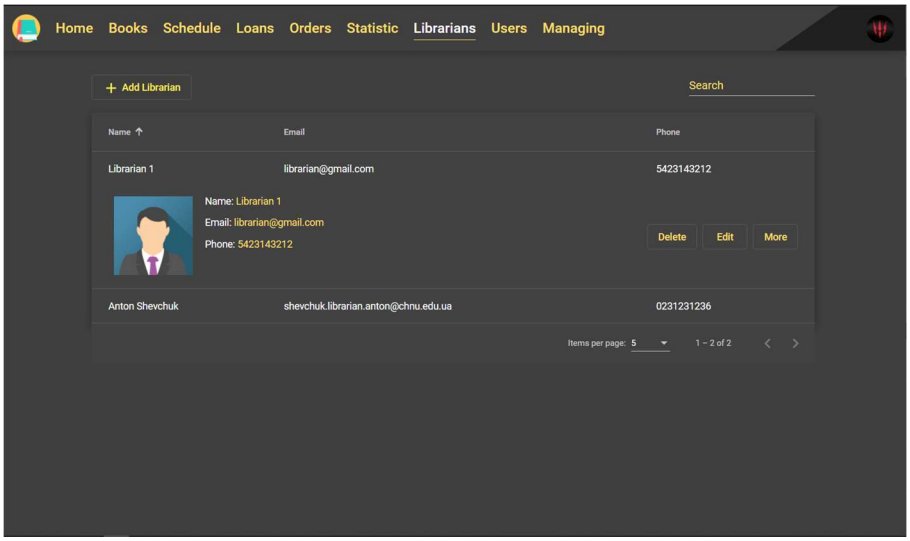


Рис. 3.48. Список бібліотекарів

Адміністратор може видаляти, редагувати, добавляти бібліотекарів. Також він може перейти на сторінку бібліотекаря (Рис. 3.49).

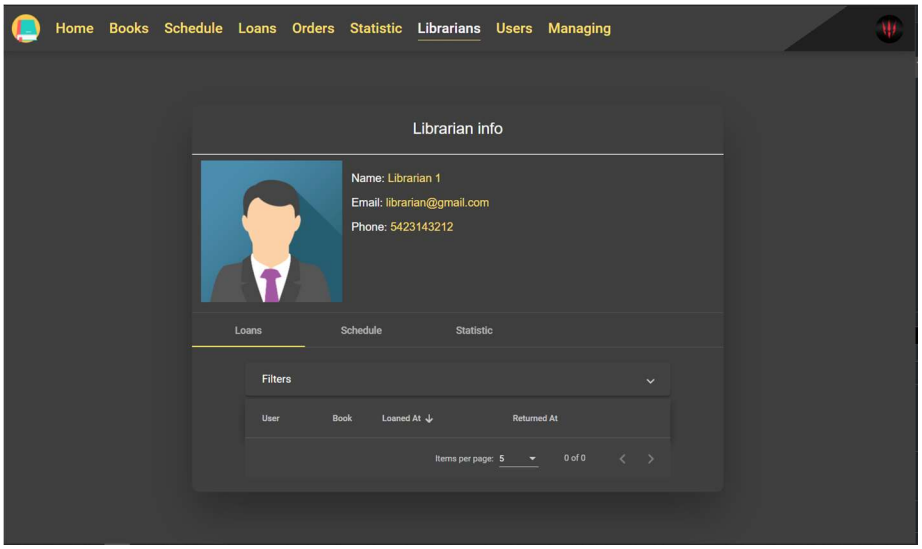
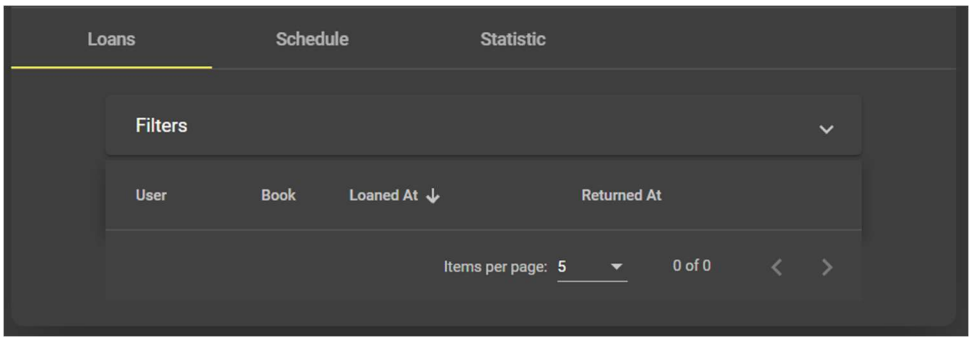


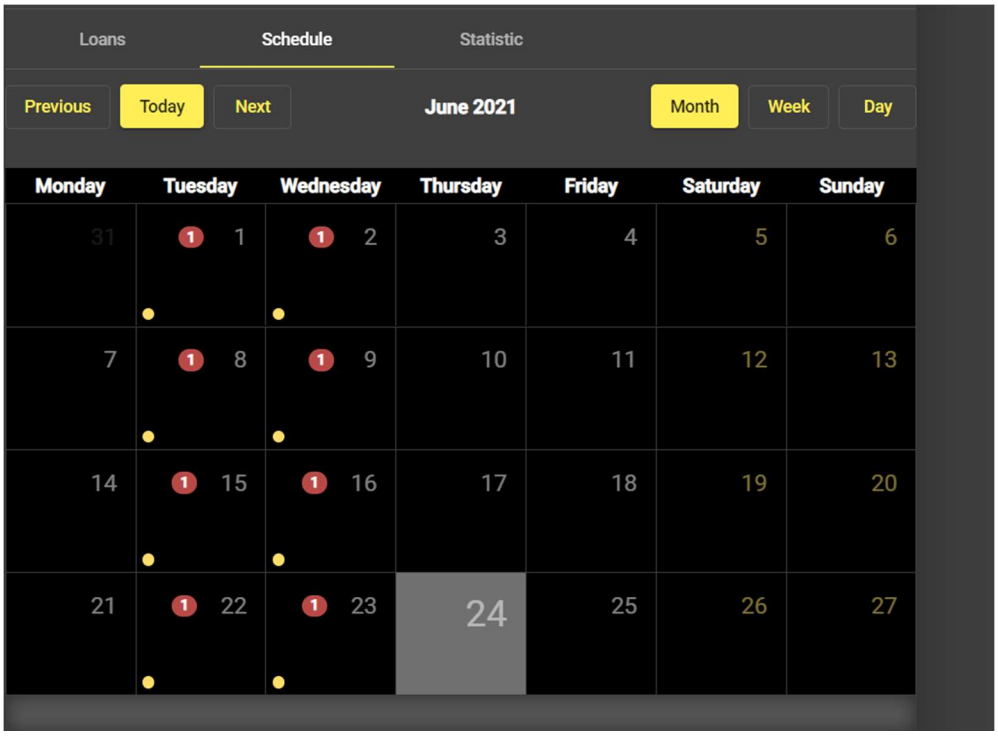
Рис. 3.49. Сторінка бібліотекаря

На сторінці бібліотекаря крім основної інформації адміністратор може переглядати інформацію про позики бібліотекаря, його розклад та статистику (рис. 3.50).

а)



б)



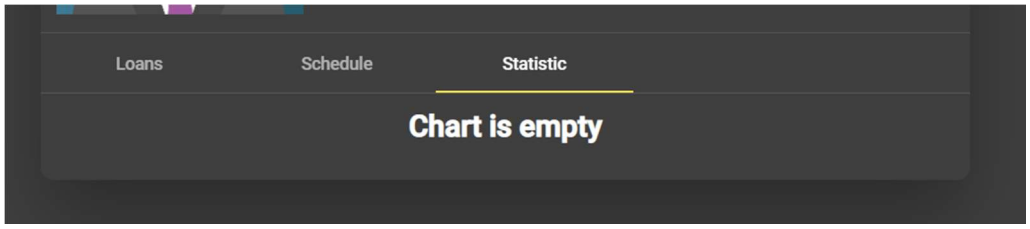
в) 

Рис. 3.50. Додаткова інформація про бібліотекаря: а) Позики; б) Розклад; в) Статистика

На сторінці управління адміністратор має доступ до перегляду розкладів, редагування, видалення, додавання їх (рис. 3.51).

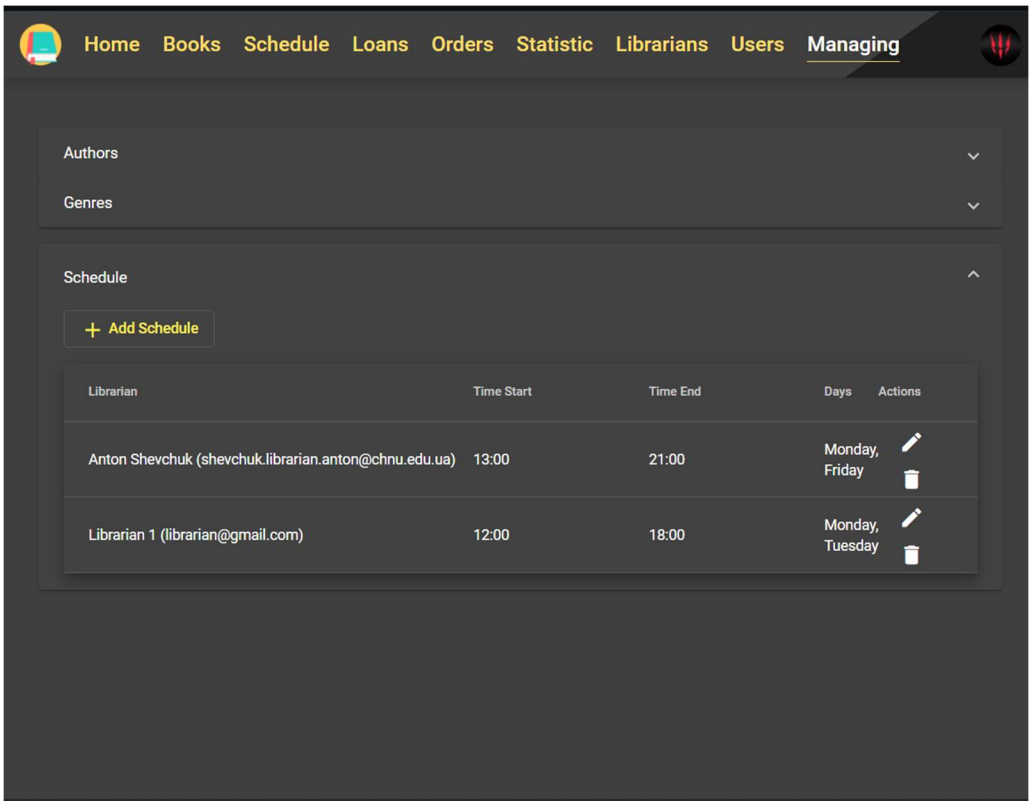


Рис. 3.51. Сторінка управління

Висновки до розділу 3

В третьому розділі документації описуються загальні відомості про програму, подається алгоритм роботи. Показується структура програмного забезпечення, діаграма прецедентів, діаграма бази даних створеного програмного додатку та описується функціонал програми.

ВИСНОВКИ

Результатом виконання дипломної роботи є розроблений веб-додаток «Бібліотека» для оптимізації роботи з каталогами книг.

Актуальність створеного програмного забезпечення полягає в вирішенні проблеми дистанційного замовлення та отримання книги у бібліотеці, в оптимізації управління бібліотекою та в створенні зрозумілого і простого інтерфейсу для користувачів.

Програмна розробка виконана з використанням платформи Node.js та каркасу Express.js на серверній частині, з використанням фреймворку Angular на клієнтській частині, з використанням мови програмування TypeScript на обох частинах веб-додатку. Як ДОСУБД було обрано MongoDB.

Розроблений веб-додаток автоматизує роботу адміністраторів та бібліотекарів, полегшує отримання книг користувачам без перебування в чергах і витрачання зайвого часу. Програму можна рекомендувати для застосування в бібліотеках будь-якої величини.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліотечні ресурси. URL: http://lib.rada.gov.ua/static/about/wellcome.html.
2. Національний університет "Києво-Могилянська академія". Головна. URL: https://library.ukma.edu.ua/.
3. Національна бібліотека України імені В. І. Вернадського. URL: http://www.nbuv.gov.ua/.
4. Національна бібліотека України ім. Ярослава Мудрого. URL: https://nlu.org.ua/.
5. Про затвердження Концепції формування системи національних електронних інформаційних ресурсів | Національна бібліотека України імені В. І. Вернадського. URL: http://nbuv.gov.ua/node/1414.
6. "Створення і функціонування електронних бібліотечних систем як чинник розвитку єдиної національної освітньо-наукової системи". Аналітична записка. Національний інститут стратегічних досліджень. URL: https://niss.gov.ua/doslidzhennya/gumanitarniy-rozvitok/stvorennya-i-funkcionuvannya-elektronnikh-bibliotechnikh-sistem.
7. Academic Publishing | Academic books, ebooks, reference books and textbooks | Cambridge University Press. Cambridge University Press. URL: https://www.cambridge.org/ua/academic.
8. Angular. URL: https://angular.io/docs
9. Directory of Open Access Journals. Directory of Open Access Journals – DOAJ. URL: https://www.doaj.org/.
10. Express/Node introduction - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.
11. Holmes S. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2015. 440 p.
12. Library. URL: https://library.te.ua/wp-content/uploads/2011/10/pubdir03.pdf.

13. Oxford Reference - Answers with Authority. Oxford Reference. URL: https://www.oxfordreference.com.
14. ScienceDirect.com | Science, health and medical journals, full text articles and books. ScienceDirect.com | Science, health and medical journals, full text articles and books. URL: https://www.sciencedirect.com/.
15. The most popular database for modern apps. MongoDB. URL: https://www.mongodb.com/.
16. UC Berkeley Library. URL: https://www.lib.berkeley.edu/.
17. Universitätsbibliothek Regensburg - Universität Regensburg. Home - UniversitÃ¤t Regensburg. URL: https://www.uni-regensburg.de/bibliothek/startseite/index.html.
18. Vanderkam D. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. O'Reilly Media, Incorporated, 2019. 250 p.

ДОДАТКИ

Додаток А. Код програми

1. **/src/index.ts**

```
// Головний файл
import express, { Request, Response } from 'express';
import bodyParser from 'body-parser';
import passport from 'passport';
import { config } from 'dotenv';
import path from 'path';

import cors from './config/cors';
import connectMongoDB from './config/db';
import logger from './config/logger';
import multer from './config/multer';
import usePassport from './config/passport';

import useRoutes from './routes/index';

if (process.env.NODE_ENV !== 'production') {
   config();
}

const app = express();
const port = process.env.PORT || 3000;
// Ініціалізація проміжного програмного забезпечення для авторизації та
// аутентифікаціїї
app.use(passport.initialize());
app.use(passport.session());

app.use(bodyParser.json({ limit: '10mb' }));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(multer());
app.use(express.static(path.join(__dirname, 'public')));
app.use('assets', express.static(path.join(__dirname, 'assets')));
```

```
app.use(cors);

if (process.env.NODE_ENV === 'production') {
  app.use(express.static('angular/dist/angular'));
}

usePassport(passport);
useRoutes(app);

app.get('*', (req: Request, res: Response) => {
  res.sendFile(path.join(__dirname + '/../angular/dist/angular/index.html'));
});

(async () => {
  try {
    // Підключення до ДБ
    await connectMongoDB();
    // Запуск сервера
    app.listen(port);
    logger.info('Successfully connected to MongoDB');
    logger.info('App is listening on', port);
  } catch (err) {
    logger.error('Cannot connect to MongoDB. Error:', err.message);
  }
})();
```

2. **/src/config/passport.ts**

```
import * as passportJWT from 'passport-jwt';
import { Strategy as LocalStrategy } from 'passport-local';
import { Request } from 'express';

import logger from './logger';

import User from '../schemas/user';
```

```
import { UserSchema } from '../models/user';

import fields from '../constants/fields';
import errorMessages from '../constants/errorMessages';

const { SECRET_KEY } = process.env;
const JWTStrategy = passportJWT.Strategy;
const ExtractJWT = passportJWT.ExtractJwt;

export default (passport: any) => {
  passport.use(
    new LocalStrategy(
      {
        usernameField: fields.EMAIL,
        passwordField: fields.PASSWORD,
        passReqToCallback: true
      },
      async (req: Request, email: string, password: string, done: any) => {

        try {
          logger.info(`Login attempt ${email}`);
          const user = await User.findOne({ email }) as UserSchema;

          if (!user) {
            logger.warn(`User ${email} does not exist`);
            return done(null, false, { message:
errorMessages.INCORRECT_LOGIN_DATA});
          }

          if (!user.active) {
            logger.warn(`User ${email} is not active`);
            return done(null, false, { message: errorMessages.NOT_ACTIVE });
          }

          if (!(await user.comparePassword(password))) {
```

```
                logger.warn(`User ${email} incorrect data`);
                return done(null, false, { message:
errorMessages.INCORRECT_LOGIN_DATA });
            }

            return done(null, user);
          } catch (err) {
            logger.error(`Authentication error: ${err.message}`);
            return done(null, false, { message:
errorMessages.SOMETHING_WENT_WRONG });
          }
        }
      )
    );

    // Для використання JWT токена
    passport.use(
      new JWTStrategy({
          jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
          secretOrKey: SECRET_KEY
        },
        async (jwtPayload, cb) => {
          try {
            const user = await User.findById(jwtPayload.id);
            return cb(null, user);
          } catch (err) {
            return cb(err);
          }
        }
      )
    );

    passport.serializeUser((auth: any, done: any) => done(null, auth.id));

    passport.deserializeUser(async (id: string, done: any) => {
```

```
        try {
            const user = await User.findById(id);
            done(null, user);
        } catch (err) {
            done(err);
        }
    });
};
```

3. **/src/controllers/book.ts**

```
import { Request, Response } from 'express';
import moment from 'moment';

import Book from '../schemas/book';
import Loan from '../schemas/loan';

import { BookSchema } from '../models/book';

import { responseErrorHandle, responseSuccessHandle } from '../helper/response';
import { removedEmptyFields } from '../helper/object';
import {uploadFileToStorage, uploadImageToStorage} from '../helper/storage';

import logger from '../config/logger';

import errorMessages from '../constants/errorMessages';
import successMessages from '../constants/successMessages';
// Для витягування книг по фільтру та пагінації
export const getBooks = async (req: Request, res: Response) => {
    const { filterValue, yFrom, yTo, pageSize, language } = req.query;
    const page = Number(req.query.page);
    const authors = String(req.query.authors).split(',');
    const genres = String(req.query.genres).split(',');
    const onlyEbooks = !!req.query.onlyEbooks;
    const onlyNormalBooks = !!req.query.onlyNormalBooks;
```

```
    const regex = new RegExp(filterValue as string, 'i');
    // Об’єкт на основі якого відбувається фільтр
    const filterCondition: any = {
      quantity: { $gt: 0 },
      $and: [ { $or: [{title: regex }, { isbn: regex }, { description: regex } ] } ]
    };
    const filter: any = removedEmptyFields(filterCondition);
    authors.forEach(author => filter.$and = filter.$and.concat(removedEmptyFields({
'authors.author': author })));
    genres.forEach(genre => filter.$and = filter.$and.concat(removedEmptyFields({
'genres.genre': genre })));

    if (language) {
      filter.$and = filter.$and.concat({ language });
    }

    if (onlyEbooks || onlyNormalBooks) {
      filter.$and = filter.$and.concat({ ebook: onlyEbooks || { $ne: true } });
    }

    if (yFrom || yTo) {
      filter.year = {
        $gt: Number(yFrom) - 1 || 0,
        $lt: Number(yTo) + 1 || new Date().getFullYear() + 1,
      };
    }

    try {
      const length = await Book.countDocuments(filter);
      const books = await Book
        .find(filter, {}, {
          limit: Number(pageSize),
          skip: Number(page) * Number(pageSize),
          sort: { year: -1 }
        })
```

```
            .populate('authors.author')
            .populate('genres.genre') as BookSchema[];
        const booksData = books.map(book => ({
            ...book.toJSON(),
            authors: book.authors.map(author => author.author),
            genres: book.genres.map(genre => genre.genre),
        }));
        const data = {
            books: booksData,
            message: successMessages.SUCCESSFULLY_FETCHED,
            pagination: { page, length }
        };
        responseSuccessHandle(res, 200, data);
    } catch (err) {
        logger.error(`Error fetching books: ${err.message}`);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для витягування книги
export const getBook = async (req: Request, res: Response) => {
    const id = req.params.id;

    if (!id) {
        return responseErrorHandle(res, 500,
errorMessages.SOMETHING_WENT_WRONG);
    }

    try {
        const book = await
Book.findById(id).populate('authors.author').populate('genres.genre') as BookSchema;
        const bookData = {
            ...book.toJSON(),
            authors: book.authors.map(author => author.author),
            genres: book.genres.map(genre => genre.genre),
        };
```

```
      responseSuccessHandle(res, 200, { book: bookData, message:
successMessages.SUCCESSFULLY_FETCHED });
   } catch (err) {
      responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
   }
};
// Для витягування статистики книги
export const getBookStats = async (req: Request, res: Response) => {
   const id = req.params.id;
   const monthAgo = moment().subtract('1', 'months').toDate();

   try {
      const loansAllTime = await Loan.countDocuments({ book: id });
      const loansForLastMonth = await Loan.countDocuments({  loanedAt: { $gt:
monthAgo }, book: id });
      const notReturnedBooks = await Loan.countDocuments({ book: id, returnedAt: {
$exists: false } });
      const bookStats = { loansForLastMonth, loansAllTime, notReturnedBooks };
      responseSuccessHandle(res, 200, { bookStats, message:
successMessages.SUCCESSFULLY_FETCHED });
   } catch (err) {
      responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
   }
};
// Для додавання книги
export const addBook = async (req: Request, res: Response) => {
   const book = JSON.parse(req.body.book);
   const file = req.file;

   if (!book) {
      return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
   }

   try {
      const isExists = await Book.findOne({ isbn: book.isbn });
```

```
        if (isExists) {
            return responseErrorHandle(res, 400, errorMessages.ISBN_EXIST);
        }

        if (book.ebook) {
            book.file = await uploadFileToStorage(file.path);
        }

        book.image = await uploadImageToStorage(book.image);
        book.genres = book.genres.map((genre: any) => ({ genre: genre.id }));
        book.authors = book.authors.map((author: any) => ({ author: author.id }));
        await Book.create(book);
        responseSuccessHandle(res, 200, { message:
successMessages.BOOK_SUCCESSFULLY_CREATED });
    } catch (err) {
        logger.error('Error creating book', err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для редагування книги
export const editBook = async (req: Request, res: Response) => {
    const id = req.params.id;
    const book = JSON.parse(req.body.book);

    if (!book) {
        return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
    }

    try {
        const isExists = await Book.findOne({ _id: { $ne: id }, isbn: book.isbn });

        if (isExists) {
            return responseErrorHandle(res, 400, errorMessages.ISBN_EXIST);
        }
```

```
        book.genres = book.genres.map((genre: any) => ({ genre: genre.id }));
        book.authors = book.authors.map((author: any) => ({ author: author.id }));
        await Book.findByIdAndUpdate(id, book);
        responseSuccessHandle(res, 200, { message:
successMessages.BOOK_SUCCESSFULLY_UPDATED });
    } catch (err) {
        logger.error(`Error updating book`, err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для видалення книги
export const deleteBook = async (req: Request, res: Response) => {
    const { id } = req.params;

    try {
        await Book.findByIdAndDelete(id);
        responseSuccessHandle(res, 200, { message:
successMessages.BOOK_SUCCESSFULLY_DELETED });
    } catch (err) {
        logger.error(`Error deleting book: ${err.message}`);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
```

4. **/src/controllers/loans**

```
import { Request, Response } from 'express';
import moment from 'moment';

import logger from '../config/logger';

import Book from '../schemas/book';
import Loan from '../schemas/loan';
import User from '../schemas/user';
```

```
import { LoanSchema, Statistic } from '../models/loan';
import { BookSchema } from '../models/book';
import { UserSchema } from '../models/user';

import { responseErrorHandle, responseSuccessHandle } from '../helper/response';
import { removedEmptyFields } from '../helper/object';

import errorMessages from '../constants/errorMessages';
import successMessages from '../constants/successMessages';
import { getStatistic } from '../helper/statistic';
// Для витягування позик
export const getLoans = async (req: Request, res: Response) => {
   const { sortName, sortOrder, page, pageSize, loanedAt, librarianId, userId } =
req.query;
   const showOnlyDebtors = !!req.query.showOnlyDebtors;
   const showOnlyReturned = !!req.query.showOnlyReturned;

   const sort: any = {};
   sort[sortName as string] = sortOrder;
   const filterDate = new Date(String(loanedAt));
   const tomorrow = moment(filterDate.getTime()).add(1, 'days').toDate();
   const filterCondition = {
      loanedAt: loanedAt && { $gte: filterDate, $lt: tomorrow },
      returnedAt: (showOnlyReturned && { $exists: true, $ne: null }) ||
(showOnlyDebtors && { $exists: false }) || null
   };
   const filter: any = removedEmptyFields(filterCondition);

   if (librarianId) {
      filter.$and = [ { librarian: librarianId } ];
   }

   if (userId) {
      filter.$and = [ { user: userId } ];
   }
```

```
    try {
        const quantity = await Loan.countDocuments(filter);
        const loans = await Loan
            .find(filter, {}, { limit: Number(pageSize), skip: Number(page) *
Number(pageSize), sort })
            .populate('book')
            .populate('user')
            .populate('librarian') as LoanSchema[];
        const loansData = loans.map(loan => {
            const { user, book, librarian } = loan;

            return ({
                ...loan.toJSON(),
                user: { name: user.name, email: user.email, phone: user.phone },
                book: { title: book.title, isbn: book.isbn, year: book.year },
                librarian: { name: librarian.name, email: librarian.email, phone: librarian.phone
},
            });
        });
        const data = { loans: loansData, quantity, message:
successMessages.SUCCESSFULLY_FETCHED };

        return responseSuccessHandle(res, 200, data);
    } catch (err) {
        logger.error(`Error getting loans: ${err.message}`);
        return responseErrorHandle(res, 500, errorMessages.CANNOT_FETCH);
    }
};
// Для повернення кнги до бібліотеки
export const returnBook = async (req: Request, res: Response) => {
    const { id } = req.params;

    if (!id) {
```

```
        return responseErrorHandle(res, 500,
errorMessages.SOMETHING_WENT_WRONG);
    }

    try {
        const loan = await Loan.findById(id) as LoanSchema;

        if (!loan) {
            return responseErrorHandle(res, 500,
errorMessages.SOMETHING_WENT_WRONG);
        }

        const book = await Book.findById(loan.book) as BookSchema;
        await Book.findByIdAndUpdate(loan.book, { quantity: book.quantity + 1 });
        await Loan.findByIdAndUpdate(id, { returnedAt: new Date() });
        responseSuccessHandle(res, 200, { message:
successMessages.SUCCESSFULLY_RETURNED_BOOK });
    } catch (err) {
        logger.error(`Error returning book: ${err.message}`);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для взяття книги
export const loanBook = async (req: Request, res: Response) => {
    const { userCredentials, librarianId, bookId } = req.body;

    if (!userCredentials || !librarianId || !bookId) {
        return responseErrorHandle(res, 400,
errorMessages.SOMETHING_WENT_WRONG);
    }

    try {
        const student = { librarian: { $ne: true }, admin: { $ne: true } };
        const user = await User.findOne({ ...student, $or: [ { phone: userCredentials }, {
email: userCredentials } ] });
```

```
      if (!user) {
        return responseErrorHandle(res, 400, errorMessages.CANNOT_FIND_USER);
      }

      const book = await Book.findOne({ _id: bookId, quantity: { $gt: 0 } }) as
BookSchema;

      if (!book) {
        return responseErrorHandle(res, 500,
errorMessages.BOOK_IS_NOT_AVAILABLE_NOW);
      }

      await Loan.create({ book: bookId, user: user._id, librarian: librarianId, loanedAt:
new Date() });
      await Book.findByIdAndUpdate(bookId, { quantity: book.quantity - 1 });
      responseSuccessHandle(res, 200, { message:
successMessages.SUCCESSFULLY_LOANED });
    } catch (err) {
      logger.error(`Error loaning book: ${err.message}`);
      responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для витягування статистики користувача
export const getUserStatistic = async (req: Request, res: Response) => {
    try {
      const { email } = req.query;
      const monthAgo = moment().subtract('1', 'months').toDate();
      const user = await User.findOne({ email }) as UserSchema;
      const loans = await Loan.find({  loanedAt: { $gt: monthAgo }, user: user._id }) as
LoanSchema[];
      const statistic: Statistic[] = getStatistic(loans);
      responseSuccessHandle(res, 200,  { message:
successMessages.SUCCESSFULLY_FETCHED, statistic });
    } catch (err) {
```

```
        logger.error('Error getting summary statistic', err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для витягування статистики бібліотекаря
export const getLibrarianStatistic = async (req: Request, res: Response) => {
    try {
        const { email } = req.query;
        const monthAgo = moment().subtract('1', 'months').toDate();
        const librarian = await User.findOne({ email }) as UserSchema;
        const loans = await Loan.find({  loanedAt: { $gt: monthAgo }, librarian:
librarian._id }) as LoanSchema[];
        const statistic: Statistic[] = getStatistic(loans);
        responseSuccessHandle(res, 200,  { message:
successMessages.SUCCESSFULLY_FETCHED, statistic });
    } catch (err) {
        logger.error('Error getting summary statistic', err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для витягування статистики книги
export const getBookStatistic = async (req: Request, res: Response) => {
    try {
        const { isbn } = req.query;
        const monthAgo = moment().subtract('1', 'months').toDate();
        const book = await Book.findOne({ isbn }) as BookSchema;

        if (!book) {
            return responseErrorHandle(res, 500,
errorMessages.SOMETHING_WENT_WRONG);
        }

        const loans = await Loan.find({  loanedAt: { $gt: monthAgo }, book: book._id }) as
LoanSchema[];
        const statistic: Statistic[] = getStatistic(loans);
```

```
        responseSuccessHandle(res, 200,  { message:
successMessages.SUCCESSFULLY_FETCHED, statistic });
    } catch (err) {
        logger.error('Error getting summary statistic', err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для повернення загальної статистики
export const getSummaryStatistic = async (req: Request, res: Response) => {
    try {
        const monthAgo = moment().subtract('1', 'months').toDate();
        const totalBooks = await Book.countDocuments();
        const loansForLastMonth = await Loan.countDocuments({  loanedAt: { $gt:
monthAgo } });
        const allDebtorsLoans = await Loan.find({  loanedAt: { $gt: monthAgo },
returnedAt: { $exists: false } }) as LoanSchema[];
        const totalDebtors = new Set(allDebtorsLoans.map(loan => loan.user)).size;
        const data = {
            summaryStatistic: { totalBooks, loansForLastMonth, totalDebtors },
            message: successMessages.SUCCESSFULLY_FETCHED
        };
        responseSuccessHandle(res, 200,  data);
    } catch (err) {
        logger.error('Error getting summary statistic', err.message);
        responseErrorHandle(res, 500, errorMessages.SOMETHING_WENT_WRONG);
    }
};
```

5. **/src/controllers/user.ts**

```
import { Request, Response } from 'express';

import logger from '../config/logger';

import User from '../schemas/user';
```

```
import { UserSchema } from '../models/user';

import successMessages from '../constants/successMessages';
import errorMessages from '../constants/errorMessages';
import { emailSubjects, generateUserCreationMessage } from '../constants/email';

import { responseErrorHandle, responseSuccessHandle } from '../helper/response';
import { generatePassword } from '../helper/password';
import { sendMail } from '../helper/email';
import { uploadImageToStorage } from '../helper/storage';
// Для створення користувача
export const createUser = async (req: Request, res: Response) => {
  const { name, email, phone, admin, librarian } = req.body;
  const password = generatePassword();

  if (!name || !email || !phone) {
    return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
  }

  const isUserWithEmailExists = !!(await User.findOne({ email }));

  if (isUserWithEmailExists) {
    return responseErrorHandle(res, 400, errorMessages.USER_EMAIL_EXISTS);
  }

  const isUserWithPhoneExists = !!(await User.findOne({ phone }));

  if (isUserWithPhoneExists) {
    return responseErrorHandle(res, 400, errorMessages.USER_PHONE_EXISTS);
  }

  try {
    await User.create({ name, email, phone, password, admin: admin || false, librarian:
librarian || false, active: true });
```

```
        await sendMail(email, emailSubjects.ACCOUNT_CREATED,
generateUserCreationMessage(email, password));
        responseSuccessHandle(res, 200, { message:
successMessages.USER_SUCCESSFULLY_CREATED });
    } catch (err) {
        logger.error('Error creating user', err.message);
        return responseErrorHandle(res, 400,
errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для редагування користувача
export const editUser = async (req: Request, res: Response) => {
    const id = req.params.id;
    const { name, email, phone } = req.body;

    if (!name || !email || !phone) {
        return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
    }

    try {
        await User.findByIdAndUpdate(id, { name, email, phone });
        responseSuccessHandle(res, 200, { message:
successMessages.USER_SUCCESSFULLY_UPDATED });
    } catch (err) {
        logger.error('Error updating user', err.message);
        return responseErrorHandle(res, 400,
errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для зміни пароля
export const editPassword = async (req: Request, res: Response) => {
    const id = req.params.id;
    const { oldPassword, newPassword } = req.body;

    if (!oldPassword || !newPassword) {
```

```
        return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
    }

    try {
        const user = await User.findById(id) as UserSchema;

        if (!user) {
            return responseErrorHandle(res, 400,
errorMessages.USER_DOES_NOT_EXIST);
        }

        if (!(await user.comparePassword(oldPassword))) {
            return responseErrorHandle(res, 400,
errorMessages.WRONG_OLD_PASSWORD);
        }

        if (await user.comparePassword(newPassword)) {
            return responseErrorHandle(res, 400,
errorMessages.OLD_PASSWORD_EQUEL_NEW_PASSWORD);
        }

        user.password = newPassword;
        await user.save();

        responseSuccessHandle(res, 200, { message:
successMessages.PASSWORD_SUCCESSFULLY_UPDATED });
    } catch (err) {
        logger.error('Error updating user', err.message);
        return responseErrorHandle(res, 400,
errorMessages.PASSWORD_ERROR_CHANGED);
    }
};
// Для зміни зображення
export const editImage = async (req: Request, res: Response) => {
    const id = req.params.id;
```

```
    const { image } = JSON.parse(req.body.user);

    if (!image) {
        return responseErrorHandle(res, 400, errorMessages.EMPTY_FIELDS);
    }

    try {
        const user = await User.findById(id) as UserSchema;

        if (!user) {
            return responseErrorHandle(res, 400,
errorMessages.USER_DOES_NOT_EXIST);
        }

        user.image = await uploadImageToStorage(image);
        await user.save();

        responseSuccessHandle(res, 200, { message:
successMessages.IMAGE_SUCCESSFULLY_UPDATED });
    } catch (err) {
        logger.error('Error updating user', err.message);
        responseErrorHandle(res, 400, errorMessages.SOMETHING_WENT_WRONG);
    }
};
// Для видалення користувача
export const deleteUser = async (req: Request, res: Response) => {
    const id = req.params.id;

    try {
        await User.findByIdAndDelete(id);
        responseSuccessHandle(res, 200, { message:
successMessages.USER_SUCCESSFULLY_DELETED });
    } catch (err) {
        logger.error('Error deleting user', err.message);
        responseErrorHandle(res, 400, errorMessages.SOMETHING_WENT_WRONG);
```

```
    }
};
// Для витягуваня користувача
export const getUser = async (req: Request, res: Response) => {
  const { id } = req.query;

  try {
     const user = await User.findById(id) as UserSchema;

     if (!user) {
        return responseErrorHandle(res, 500, errorMessages.USER_DOES_NOT_EXIST);
     }

     responseSuccessHandle(res, 200, { user: { ...user.toJSON(), password: null } });
  } catch (err) {
     logger.error('Error fetching user', err.message);
     responseErrorHandle(res, 500, 'Cannot fetch user');
  }
};
```

6. **/src/routes/book.ts**

```
import passport from 'passport';
import express from 'express';

import {addBook, deleteBook, editBook, getBook, getBooks, getBookStats} from
'../controllers/book';

const router = express.Router();
// Налаштування API
router.get('', getBooks);
router.get('/:id', getBook);
router.get('/:id/stats', getBookStats);
router.post('', passport.authenticate('jwt', { session: false }), addBook);
router.put('/:id', passport.authenticate('jwt', { session: false }), editBook);
router.delete('/:id', passport.authenticate('jwt', { session: false }), deleteBook);
```

```
export default router;
```

7. **/src/schemas/book.ts**

```
import * as mongoose from 'mongoose';
import { Schema } from 'mongoose';

const bookSchema: Schema = new Schema({
   isbn: { type: String, required: true },
   title: { type: String, required: true},
   year: { type: Number, required: true },
   quantity: { type: Number, required: true },
   description: { type: String, required: true },
   image: { type: String, required: true },
   language: { type: String, required: true },
   ebook: { type: Boolean, required: false },
   file: { type: String, required: false },
   authors: [ { author: { type: Schema.Types.ObjectId, ref: 'Author', required: true } } ],
   genres: [ { genre: { type: Schema.Types.ObjectId, ref: 'Genre', required: true } } ]
});

export default mongoose.model('Book', bookSchema);
```

8. **/angular/scr/app/app.module.ts**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { HTTP_INTERCEPTORS, HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { SharedModule } from '@shared/shared.module';
import { AuthModule } from './containers/auth/auth.module';

import { AppComponent } from './app.component';
```

```
import { ErrorPageComponent } from './components/error-page/error-page.component';
import { HeaderComponent } from './components/header/header.component';
import { EditPageComponent } from './components/edit-page/edit-page.component';
import { AuthorSectionComponent } from './components/edit-page/author-
section/author-section.component';
import { BookSectionComponent } from './components/edit-page/book-section/book-
section.component';
import { GenreSectionComponent } from './components/edit-page/genre-section/genre-
section.component';

import { AuthInterceptor } from './interceptors/auth.interceptor';

import { MainPageModule } from './containers/main-page/main-page.module';
import { UsersModule } from './containers/user/users.module';
import { LoansModule } from './containers/loans/loans.module';
import { LibrariansModule } from './containers/librarians/librarians.module';
import { StudentsModule } from './containers/students/students.module';
import { ScheduleSectionComponent } from './components/edit-page/schedule-
section/schedule-section.component';
import { MyOrdersModalComponent } from './components/header/my-orders-modal/my-
orders-modal.component';
import { NgxsModule, NoopNgxsExecutionStrategy } from '@ngxs/store';
import { UserState } from './store/state/user.state';
import { environment } from '../environments/environment';
import { StudentState } from './store/state/student.state';
import { MAT_AUTOCOMPLETE_SCROLL_STRATEGY } from
'@angular/material/autocomplete';
import { CloseScrollStrategy, Overlay } from '@angular/cdk/overlay';
import { LibrarianState } from './store/state/librarian.state';
import { HttpErrorInterceptor } from './interceptors/error.interceptor';
import { GenreState } from './store/state/genre.state';
import { AuthorState } from './store/state/author.state';
import { AuthorPopupComponent } from './components/popups/author-popup/author-
popup.component';
```

```
import { GenrePopupComponent } from './components/popups/genre-popup/genre-popup.component';
import { BookPopupComponent } from './components/popups/book-popup/book-popup.component';
import { BookState } from './store/state/book.state';
import { DisableFormControlDirective } from './directives/disableFormControl.directive';
import { LocalizationState } from './store/state/localization.state';
import { ScheduleState } from './store/state/schedule.state';
import { ReadPopupComponent } from './components/popups/read-popup/read-popup.component';
import { SchedulePopupComponent } from './components/popups/schedule-popup/schedule-popup.component';
import { HomeComponent } from './components/home/home.component';

export function scrollFactory(overlay: Overlay): () => CloseScrollStrategy {
  return () => overlay.scrollStrategies.close();
}

@NgModule({
  declarations: [
    AppComponent,
    ErrorPageComponent,
    HeaderComponent,
    EditPageComponent,
    AuthorSectionComponent,
    BookSectionComponent,
    GenreSectionComponent,
    ScheduleSectionComponent,
    MyOrdersModalComponent,
    AuthorPopupComponent,
    GenrePopupComponent,
    BookPopupComponent,
    DisableFormControlDirective,
    ReadPopupComponent,
    SchedulePopupComponent,
```

```
    HomeComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule,
    FormsModule,
    SharedModule,
    AuthModule,
    UsersModule,
    MainPageModule,
    LoansModule,
    LibrariansModule,
    StudentsModule,
    AppRoutingModule,
    NgxsModule.forRoot([
      UserState,
      StudentState,
      LibrarianState,
      GenreState,
      AuthorState,
      BookState, LocalizationState,
      ScheduleState
    ], {
      executionStrategy: NoopNgxsExecutionStrategy,
      developmentMode: !environment.production
    })
  ],
  providers: [
    {
      provide: MAT_AUTOCOMPLETE_SCROLL_STRATEGY,
      useFactory: scrollFactory,
      deps: [Overlay]
    },
    {
```

```
            provide: HTTP_INTERCEPTORS,
            useClass: AuthInterceptor,
            multi: true
        },
        {
            provide: HTTP_INTERCEPTORS,
            useClass: HttpErrorInterceptor,
            multi: true
        }
    ],
    entryComponents: [MyOrdersModalComponent, AuthorPopupComponent,
ReadPopupComponent, SchedulePopupComponent],
    bootstrap: [AppComponent]
})
export class AppModule {
}
```

9. /angular/scr/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { ErrorPageComponent } from './components/error-page/error-page.component';
import { EditPageComponent } from './components/edit-page/edit-page.component';

import { AngularLinks } from './constants/angularLinks';

import { AuthGuard } from './guards/auth.guard';
import { LibrarianGuard } from './guards/librarian.guard';
import { HomeComponent } from './components/home/home.component';

const routes: Routes = [
    {
        path: AngularLinks.HOME,
        component: HomeComponent,
```

```
    pathMatch: 'full'
  },
  {
    path: AngularLinks.EDIT_PAGE,
    component: EditPageComponent,
    canActivate: [AuthGuard, LibrarianGuard]
  },
  { path: '**', component: ErrorPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes, { relativeLinkResolution: 'legacy' })],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

10. /angular/scr/app/containers/librarians/ librarians / librarians.component.html

```
<section class="table-section">
  <section class="d-flex justify-content-between align-items-center mb-1">
    <button
        mat-stroked-button
        color="primary"
        (click)="onOpenCreatePopup()"
    >
      <i class="material-icons">add</i> Add Librarian
    </button>
    <mat-form-field>
      <mat-label>Search</mat-label>
      <input
          matInput
          [(ngModel)]="filterValue"
          (ngModelChange)="loadLibrariansPage()"
      />
    </mat-form-field>
  </section>
```

```
<div class="spinner-container" *ngIf="dataSource.loading$ | async">
  <mat-spinner></mat-spinner>
</div>
<table
  mat-table
  multiTemplateDataRows
  class="mat-elevation-z8"
  matSort
  matSortActive="name"
  matSortDirection="asc"
  matSortDisableClear
  [dataSource]="dataSource"
>
  <ng-container [matColumnDef]="tableColumns.NAME">
    <th mat-header-cell mat-sort-header *matHeaderCellDef>Name</th>
    <td mat-cell *matCellDef="let element">
      {{ element.name ? element.name : 'Name is not provided' }}
    </td>
  </ng-container>

  <ng-container [matColumnDef]="tableColumns.EMAIL">
    <th mat-header-cell mat-sort-header *matHeaderCellDef>Email</th>
    <td mat-cell *matCellDef="let element">{{ element.email }}</td>
  </ng-container>

  <ng-container matColumnDef="phone">
    <th mat-header-cell mat-sort-header *matHeaderCellDef>Phone</th>
    <td mat-cell *matCellDef="let element">{{ element.phone }}</td>
  </ng-container>

  <ng-container matColumnDef="expandedDetail">
    <td mat-cell
      *matCellDef="let element"
      [attr.colspan]="columnsToDisplay.length"
    >
```

```
            <div
              class="user-table-expand-detail"
              [@detailExpand]=" element == expandedElement ? 'expanded' : 'collapsed'"
            >
              <div class="profile-image-container">
                <img class="profile-image"
                  src="{{ element.profileImage? element.profileImage :
'https://www.w3schools.com/w3css/img_avatar3.png'}}"
                  alt="profile image"
                />
              </div>
              <div class="info-section">
                <p>Name: <span class="text-main">{{ element.name }}</span></p>
                <p>Email: <span class="text-main">{{ element.email }}</span></p>
                <p>Phone: <span class="text-main">{{ element.phone }}</span></p>
              </div>
              <div class="d-flex user-link">
                <button
                    mat-stroked-button
                    color="primary"
                    class="user-link-button mr-2"
                    matTooltip="Delete Student"
                    (click)="onDeleteStudent(element._id)"
                >
                  Delete
                </button>
                <button
                    mat-stroked-button
                    color="primary"
                    class="user-link-button mr-2"
                    matTooltip="Edit Student"
                    (click)="onOpenEditPopup(element)"
                >
                  Edit
                </button>
```

```
                    <a
                      mat-stroked-button
                      color="primary"
                      class="user-link-button"
                      matTooltip="See More Information"
                      [routerLink]="'/' + links.LIBRARIANS + '/' + element._id"
                    >
                      More
                    </a>
                  </div>
                </div>
              </td>
            </ng-container>

            <tr mat-header-row *matHeaderRowDef="columnsToDisplay"></tr>
            <tr
              mat-row
              *matRowDef="let element; columns: columnsToDisplay"
              class="element-row"
              [class.example-expanded-row]="expandedElement === element"
              (click)=" expandedElement = expandedElement === element ? null : element"
            ></tr>
            <tr
              mat-row
              *matRowDef="let row; columns: ['expandedDetail']"
              class="detail-row"
            ></tr>
          </table>
          <mat-paginator
            [length]="getTotalItems()"
            [pageSize]="paginator.pageSize || 5"
            [pageSizeOptions]="[5, 10]"
            #paginator
          ></mat-paginator>
        </section>
```

11. /angular/scr/app/containers/librarians/ librarians / librarians.component.ts

```
import { AfterViewInit, Component, OnDestroy, OnInit, ViewChild } from
'@angular/core';
import { MatPaginator } from '@angular/material/paginator';
import { MatSort } from '@angular/material/sort';

import { merge } from 'rxjs';
import { tap } from 'rxjs/operators';

import { AngularLinks } from '../../../constants/angularLinks';
import { TableColumns } from '../../../constants/tableColumns';
import { PageTitles } from '../../../constants/pageTitles';
import { SortOrder } from '../../../constants/sortOrder';

import { LibrariansDataSource } from '../../../datasources/librarians.datasource';
import { untilDestroyed } from 'ngx-take-until-destroy';
import { TABLE_ANIMATION } from '../../../constants/animation';
import { Store } from '@ngxs/store';
import { UserPopupData } from '@shared/user-popup/user-popup.data';
import { UserPopupComponent } from '@shared/user-popup/user-popup.component';
import { DeleteUser } from '../../../store/state/user.state';
import { MatDialog } from '@angular/material/dialog';
import { User } from '../../../models/user.model';
import { LibrarianState } from '../../../store/state/librarian.state';

@Component({
  selector: 'app-librarians',
  templateUrl: './librarians.component.html',
  animations: TABLE_ANIMATION
})
export class LibrariansComponent implements OnInit, AfterViewInit, OnDestroy {
  filterValue: string;
  links = AngularLinks;
```

```
  columnsToDisplay: string[] = [ TableColumns.NAME, TableColumns.EMAIL,
TableColumns.PHONE ];
  expandedElement: User | null;
  tableColumns = TableColumns;


  dataSource: LibrariansDataSource;
  @ViewChild(MatPaginator, { static: true }) paginator: MatPaginator;
  @ViewChild(MatSort, { static: true }) sort: MatSort;
  constructor(
    private store: Store,
    private dialog: MatDialog,
  ) {}

  ngOnInit(): void {
    document.title = PageTitles.LIBRARIANS;
    this.dataSource = new LibrariansDataSource(this.store);
    this.dataSource.loadLibrarians('', this.sort.active || 'name', SortOrder.DESC, 0,
this.paginator.pageSize || 5);
  }

  ngAfterViewInit(): void {
    this.sort.sortChange.pipe(untilDestroyed(this)).subscribe(() =>
(this.paginator.pageIndex = 0));
    merge(this.sort.sortChange,
this.paginator.page).pipe(untilDestroyed(this)).pipe(tap(() =>
this.loadLibrariansPage())).subscribe();
  }

  getTotalItems(): number {
    return this.store.selectSnapshot(LibrarianState.LibrariansTotalItems);
  }

  loadLibrariansPage(): void {
    this.dataSource
```

```
            .loadLibrarians(this.filterValue, this.sort.active, this.sort.direction,
this.paginator.pageIndex, this.paginator.pageSize);
    }

    onOpenEditPopup(user) {
        const data: UserPopupData = { user, isEdit: true };
        const dialog = this.dialog.open(UserPopupComponent, { data, width: '768px',
disableClose: true });
        dialog.afterClosed().pipe(untilDestroyed(this)).subscribe((res) => res &&
this.loadLibrariansPage());
    }

    onOpenCreatePopup() {
        const dialog = this.dialog.open(UserPopupComponent, { data: { isLibrarian: true },
width: '768px', disableClose: true });
        dialog.afterClosed().pipe(untilDestroyed(this)).subscribe((res) => res &&
this.loadLibrariansPage());
    }

    onDeleteStudent(id: string): void {
        this.store.dispatch(new DeleteUser(id)).subscribe(() => this.loadLibrariansPage());
    }

    ngOnDestroy(): void {}
}
```

12. /angular/scr/app/datasources/librarians.datasource.ts

```
import { CollectionViewer, DataSource } from '@angular/cdk/collections';

import { BehaviorSubject, Observable, of } from 'rxjs';
import { catchError, finalize } from 'rxjs/operators';
import { Store } from '@ngxs/store';
import { StoreStateModel } from '../store/models/store.model';
import { LoadLibrarians } from '../store/state/librarian.state';
import { User } from '../models/user.model';
```

```
export class LibrariansDataSource implements DataSource<User> {
  private librariansSubject = new BehaviorSubject<User[]>([]);

  private loadingSubject = new BehaviorSubject<boolean>(false);

  public loading$ = this.loadingSubject.asObservable();

  constructor(private store: Store) {}

  loadLibrarians(filterValue: string, sortName: string, sortOrder: string, pageIndex:
number, pageSize: number) {
    this.loadingSubject.next(true);
    this.store
      .dispatch(new LoadLibrarians(filterValue, sortName, sortOrder, pageIndex,
pageSize))
      .pipe(catchError(() => of([])), finalize(() => this.loadingSubject.next(false)))
      .subscribe((state: StoreStateModel) =>
this.librariansSubject.next(state?.librarian?.librarians));
  }

  connect(collectionViewer: CollectionViewer): Observable<User[]> {
    return this.librariansSubject.asObservable();
  }

  disconnect(collectionViewer: CollectionViewer): void {
    this.librariansSubject.complete();
    this.loadingSubject.complete();
  }
}
```

13. /angular/scr/app/directives/disableFormControl.directive.ts

```
import { NgControl } from '@angular/forms';
import {Directive, Input} from '@angular/core';
```

```
@Directive({
  selector: '[appDisableControl]'
})
export class DisableFormControlDirective {

  @Input()
  set appDisableControl( condition: boolean ) {
    const action = condition ? 'disable' : 'enable';
    this.ngControl.control[action]();
  }

  constructor( private ngControl: NgControl ) {}

}
```

14. /angular/scr/app/guards/librarian.guard.ts

```
import {
    ActivatedRouteSnapshot,
    CanActivate,
    Router,
    RouterStateSnapshot
} from '@angular/router';
import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { AngularLinks } from '../constants/angularLinks';
import { UserState } from '../store/state/user.state';
import { Store } from '@ngxs/store';

@Injectable({
    providedIn: 'root'
})
export class LibrarianGuard implements CanActivate {
    constructor(private store: Store, private router: Router) {}
```

```
    canActivate(
      route: ActivatedRouteSnapshot,
      state: RouterStateSnapshot
    ): Observable<boolean> | Promise<boolean> | boolean {
      const user = this.store.selectSnapshot(UserState.User);
      const isHasAccess = user.librarian || user.admin;

      if (!isHasAccess) {
        this.router.navigate([AngularLinks.HOME]);
        return isHasAccess;
      }

      return isHasAccess;
    }
  }
```

15. **/angular/scr/app/interceptors/auth.interceptor.ts**

```
  import { Injectable } from '@angular/core';

  import {
    HttpEvent,
    HttpHandler,
    HttpInterceptor,
    HttpRequest
  } from '@angular/common/http';

  import { Observable } from 'rxjs';

  import { AuthService } from '../services/auth.service';

  @Injectable()
  export class AuthInterceptor implements HttpInterceptor {
    constructor(private authService: AuthService) {}
```

```
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
      req = req.clone({ setHeaders: { Authorization: `${this.authService.getJwtToken()}`
} });

      return next.handle(req);
    }
}
```

16. /angular/scr/app/services/user.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

import { map } from 'rxjs/operators';

import { serverLink } from '../constants/serverLink';
import { UpdatePasswordPayload, UpdateUserPayload } from '../models/request/user';
import { Response } from '../models/response.model';

@Injectable({ providedIn: 'root' })
export class UserService {
  private USERS_URL = `${serverLink}/users`;

  constructor(private http: HttpClient) {}

  getUser(id: string) {
    return this.http.get(`${this.USERS_URL}?id=${id}`).pipe(map((response:
Response) => response.data));
  }

  createUser(body: UpdateUserPayload) {
    return this.http.post(`${this.USERS_URL}`, body).pipe(map((response: Response)
=> response.data));
  }

  editUser(data: { _id: string, body: UpdateUserPayload }) {
```

```
        return this.http.put(`${this.USERS_URL}/${data._id}`,
    data.body).pipe(map((response: Response) => response.data));
      }

      editPassword(data: { _id: string, body: UpdatePasswordPayload }) {
        return this.http.post(`${this.USERS_URL}/${data._id}`,
    data.body).pipe(map((response: Response) => response.data));
      }

      editImage(id: string, image: string) {
        const headers = new HttpHeaders();
        const formData: FormData = new FormData();
        headers.append('Content-Type', 'multipart/form-data');
        formData.append('user', JSON.stringify({ image }));
        return this.http.patch(`${this.USERS_URL}/${id}`, formData, { headers
    }).pipe(map((response: Response) => response.data));
      }

      deleteUser(id: string) {
        return this.http.delete(`${this.USERS_URL}/${id}`).pipe(map((response:
    Response) => response.data));
      }
    }
```

17. /angular/scr/app/store/state/user.state.ts

```
    import { Action, Selector, State, StateContext } from '@ngxs/store';
    import { UserStateModel } from '../models/user.model';
    import { User } from '../../models/user.model';
    import { AuthService } from '../../services/auth.service';
    import { tap } from 'rxjs/operators';
    import { Router } from '@angular/router';
    import { AngularLinks } from '../../constants/angularLinks';
    import { MaterialService } from '../../services/material.service';
    import { Injectable } from '@angular/core';
    import { SnackBarClasses } from '../../constants/snackBarClasses';
```

```
import { RegisterUserPayload, UpdatePasswordPayload, UpdateUserPayload } from
'../../models/request/user';
import { UserService } from '../../services/user.service';
import { StudentStateModel } from '../models/student.model';


/**********************************
 *** UserActions - Commands ***
 **********************************/
export class InitUserState {
   static readonly type = '[User] InitUserState';
}

export class Login {
   static readonly type = '[User] Login';

   constructor(public email: string, public password: string) {}
}

export class AutoLogin {
   static readonly type = '[User] AutoLogin';

   constructor() {}
}

export class Logout {
   static readonly type = '[User] Logout';

   constructor() {}
}

export class AutoLogout {
   static readonly type = '[User] AutoLogout';

   constructor(public expirationDuration: number) {}
```

```
}

export class RegisterUser {
  static readonly type = '[User] RegisterUser';

  constructor(public data: RegisterUserPayload) {}
}

export class LoadUser {
  static readonly type = '[User] LoadUser';

  constructor() {}
}

export class SetUser {
  static readonly type = '[User] SetUser';

  constructor(public user: User) {}
}

export class CreateUser {
  static readonly type = '[User] CreateUser';

  constructor(public data: UpdateUserPayload) {}
}

export class CheckActivationToken {
  static readonly type = '[User] CheckActivationToken';

  constructor(public token: string) {}
}

export class EditUser {
  static readonly type = '[User] EditUser';
```

```
    constructor(public data: UpdateUserPayload, public userId?: string) {}
}

export class EditPassword {
    static readonly type = '[User] EditPassword';

    constructor(public data: UpdatePasswordPayload) {}
}

export class EditImage {
    static readonly type = '[User] EditImage';

    constructor(public image: string) {}
}

export class DeleteUser {
    static readonly type = '[User] DeleteUser';

    constructor(public id?: string) {}
}

/*******************************
 *** UserState            ***
 *******************************/
export const STATE_NAME = 'user';

@State<UserStateModel>({
    name: STATE_NAME,
    defaults: new UserStateModel()
})

@Injectable()
export class UserState {
    constructor(
        private authService: AuthService,
```

```
    private userService: UserService,
    private materialService: MaterialService,
    private router: Router,
  ) { }

  /****************
   *** Selectors ***
   *****************/
  @Selector()
  static User(state: UserStateModel): User {
    return state.user;
  }

  /****************
   *** Resolvers ***
   *****************/
  @Action(InitUserState)
  initUserState(ctx: StateContext<UserStateModel>) {
    ctx.setState(new UserStateModel());
    return ctx;
  }

  @Action(LoadUser)
  loadUser(ctx: StateContext<UserStateModel>) {
    const { _id } = ctx.getState().user;
    return this.userService.getUser(_id).pipe(tap(async response => ctx.dispatch(new
SetUser(response.user))));
  }

  @Action(SetUser)
  setUser(ctx: StateContext<UserStateModel>, action: SetUser) {
    localStorage.setItem('userData', JSON.stringify(action.user));
    return ctx.patchState({ user: action.user });
  }
```

```
  @Action(Login)
  login(ctx: StateContext<UserStateModel>, action: Login) {
    const { email, password } = action;
    return this.authService.login(email, password).pipe(tap(async response => {
      const { token, user, tokenExpiresIn } = response;

      this.authService.setJwtToken(token);
      const expirationDate = new Date(new Date().getTime() + tokenExpiresIn *
1000);
      const tokenData = { token, expirationDate };
      ctx.dispatch(new AutoLogout(tokenExpiresIn * 1000));
      ctx.dispatch(new SetUser(user));
      localStorage.setItem('tokenData', JSON.stringify(tokenData));
      await this.router.navigate([AngularLinks.HOME]);
    }));
  }

  @Action(AutoLogin)
  autoLogin(ctx: StateContext<UserStateModel>) {
    const user = JSON.parse(localStorage.getItem('userData'));
    const tokenData: { token: string; expirationDate: string; } =
JSON.parse(localStorage.getItem('tokenData'));

    if (!user) {
      return;
    }

    if (!tokenData) {
      return;
    }

    if (tokenData.token) {
      ctx.dispatch(new SetUser(user));
      this.authService.setJwtToken(tokenData.token);
```

```
        const expirationDuration = new Date(tokenData.expirationDate).getTime() - new
Date().getTime();
        return ctx.dispatch(new AutoLogout(expirationDuration));
      }

      return ctx.dispatch(new Logout());
    }

    @Action(RegisterUser)
    registerUser(ctx: StateContext<UserStateModel>, action: RegisterUser) {
      return this.authService.createUser(action.data).pipe(tap(async response => {
        this.materialService.openSnackbar(response.message, SnackBarClasses.Success);
        await this.router.navigate([AngularLinks.LOGIN]);
      }));
    }

    @Action(CreateUser)
    createUser(ctx: StateContext<UserStateModel>, action: CreateUser) {
      return this.userService.createUser(action.data).pipe(tap(async response => {
        this.materialService.openSnackbar(response.message, SnackBarClasses.Success);
      }));
    }

    @Action(CheckActivationToken)
    checkActivationToken(ctx: StateContext<UserStateModel>, action:
CheckActivationToken) {
      return this.authService.checkActivationToken(action.token).pipe(tap(async response
=> {
        this.materialService.openSnackbar(response.message, SnackBarClasses.Success);
        await this.router.navigate([AngularLinks.LOGIN]);
      }));
    }

    @Action(EditUser)
    editUser(ctx: StateContext<UserStateModel>, action: EditUser) {
```

```
        const { data, userId } = action;
        const { _id: currentUserId } = ctx.getState().user;
        const _id = userId || currentUserId;
        return this.userService.editUser({ _id, body: data }).pipe(tap(async response => {
            const { message } = response;

            if (!userId && currentUserId) {
                ctx.dispatch(new LoadUser());
            }

            this.materialService.openSnackbar(message, SnackBarClasses.Success);
        }));
    }

    @Action(EditPassword)
    editPassword(ctx: StateContext<UserStateModel>, action: EditPassword) {
        const { _id } = ctx.getState().user;
        return this.userService.editPassword({ _id, body: action.data }).pipe(tap(async
response => {
            const { message } = response;
            this.materialService.openSnackbar(message, SnackBarClasses.Success);
        }));
    }

    @Action(EditImage)
    editImage(ctx: StateContext<UserStateModel>, action: EditImage) {
        const { _id } = ctx.getState().user;
        return this.userService.editImage(_id, action.image).pipe(tap( response => {
            ctx.dispatch(new LoadUser());
            this.materialService.openSnackbar(response.message, SnackBarClasses.Success);
        }));
    }

    @Action(DeleteUser)
    deleteStudent(ctx: StateContext<StudentStateModel>, action: DeleteUser) {
```

```
    return this.userService.deleteUser(action.id).pipe(tap(response => {
      this.materialService.openSnackbar(response.message, SnackBarClasses.Success);
    }));
  }

  @Action(Logout)
  logout(ctx: StateContext<UserStateModel>) {
    return this.authService.logout().pipe(tap(async () => {
      ctx.dispatch(new SetUser(null));
      this.authService.setJwtToken(null);
      localStorage.clear();
      await this.router.navigate([AngularLinks.LOGIN]);
    }));
  }

  @Action(AutoLogout)
  AutoLogout(ctx: StateContext<UserStateModel>, action: AutoLogout) {
    const { expirationDuration } = action;

    if (expirationDuration < 0) {
      ctx.dispatch(new Logout());
      localStorage.clear();
      return;
    }

    setTimeout(() => {
      ctx.dispatch(new Logout());
    }, expirationDuration);

    return ctx;
  }
}
```

Додаток Б. Заява-Засвідчення

Заява-Засвідчення

Автора кваліфікаційної роботи

(Студента) Шевчука Антона Петровича

Номер студентської книжки: 172816

Я заявляю, що наукова робота: Створення веб-додатку «Бібліотека» для оптимізації роботи з каталогами книг

1. Була підготовлена виключно мною, * і:

А. Не порушує авторські права третіх осіб у відповідність із законом про авторське право.

Б. Повністю або частково була використана в якості основи для отримання диплому про вищу освіту або наукового ступеня мною чи іншою особою.

2. Крім того,я заявляю, що надана мною для перевірки електронна версія роботи збігається з друкованою.

Даною заявою я підтверджую, що був,(-ла) проінформований,(-на) про права та обов'язки студента,(-ки) Університету, про правила, що стусуються перевірки оригінальності наукових робіт. Тому я заявляю, що я згоден,(-на) на обробку моїх письмових робіт у відповідності з антиплагіатними процедурами Університету, а також на архівування цих робіт в базі даних інтернет системи UNICHECK згідно з антиплагіатними правилами і процедурами Університету.

Я також свідомий,(-ма) того, що у випадку, якщо робота написана мною, за рішенням Комісії університету буде містити факти, які суперечать умовам зазначеним у цій заяві, або, якщо коеіцієнти виходять за межі гранично допустимих норм, робота буде повернута на допрацювання.

_____ _____

Дата *Підпис*

*Беручи до уваги істотний внесок з боку керівника наукової роботи.