

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

КУРСОВА РОБОТА
ПОЯСНЮВАЛЬНА ЗАПИСКА
з дисципліни “Об’єктно-орієнтоване програмування”
ОСОБИСТА БІБЛІОТЕКА

Керівник, Ст. викл.
Студент гр. ПЗПІ-22-1

Проф Бондарєв В. М.
Бездітко М. А.

Комісія:

Проф.	_____	Бондарєв Н. С.
Ст. викл.	_____	Черепанова Ю. Ю.
Ст. викл.	_____	Ляпота В. М.

Харків 2023

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ**

Кафедра: *Програмної інженерії*

Дисципліна: *Об'єктно-орієнтоване програмування*

Спеціальність: *121 Інженерія програмного забезпечення*

Освітня програма: *Програмна інженерія*

Курс 1. Група *ПЗПІ-22-1*. Семестр 2.

ЗАВДАННЯ
на курсовий проект студента
Бездітка Максима Андрійовича

1 Тема проекту: Особиста бібліотека

2 Термін здачі студентом закінченого проекту: *“16” - червня - 2023 р.*

3 Вихідні дані до проекту:

Специфікація програми, методичні вказівки до виконання курсової роботи.

4 Зміст розрахунково-пояснювальної записки:

Вступ, специфікація програми, проектна специфікація, інструкція користувача, висновки

КАЛЕНДАРНИЙ ПЛАН

<i>№</i>	<i>Назва етапу</i>	<i>Термін виконання</i>
1	Видача теми, узгодження і затвердження теми	13.02.2023 - 14.03.2023 р.
2	Формулювання вимог до програми	15-03-2023 – 30-03-2023 р.
3	Розробка зберігання та пошуку даних	01-04-2023 – 11-04-2023 р.
4	Розробка функцій авторизації, реєстрації	11-04-2023 – 01-05-2023 р.
5	Розробка функцій зберігання та завантаження даних	02-05-2023 – 08-05-2023 р.
6	Тестування і доопрацювання розробленої програмної системи.	09-05-2023 – 18-05-2023 р.
7	Оформлення пояснювальної записки, додатків, графічного матеріалу	19-05-2023 – 25-05-2023 р.
8	Захист	05.06.2023 – 16.06.2023 р.

Студент _____ Бездітко М. А.
 (Прізвище, Ім'я, По батькові)

Керівник _____ Бондарєв В. М.
 (Прізвище, Ім'я, По батькові)

«13» лютого _____ 2023 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 39 с., 22 рис., 1 додаток, 7 джерел.

ОСОБИСТА БІБЛІОТЕКА, МОВА ПРОГРАМУВАННЯ C#, КЛАС, ООП, ФРЕЙМВОРК ASP .NET CORE.

Метою роботи є розробка програми “Особиста бібліотека”, яка дозволить користувачам мати власний обліковий запис та відслідковувати книги, які той читає, записувати цитати.

В результаті отримана програма, що дозволяє зареєструватися, входити до акаунту, зберігати список книг, характеристики кожної книги, такі як: назва, автор, дата додавання, статус. Є можливість додавати нові книги до списку та видаляти їх, а також додавати цитати з прив’язкою до конкретної книги.

ЗМІСТ

Вступ	6
1 Специфікація програми	7
1.1 Функції програми	7
1.2 Інтерфейс користувача	8
2 Проектна специфікація	13
2.1 Об'єктна модель програми	13
2.2 Реалізація функцій програми	15
2.3 Формат даних	16
3 Інструкція користувача	18
Висновки	23
Перелік джерел посилання	24
Додаток А Код програми	25

ВСТУП

Робота полягає у створенні веб-додатку для відслідковування книг, які читає користувач, та для відслідковування найважливіших думок з книг: користувач має можливість оформити їх у цитати. У роботі використовуються принципи об'єктно-орієнтованого програмування.

Темою роботи є програма, яка підтримує та взаємодіє з базою даних через додавання, редагування, видалення користувацьких даних, а також здатна робити пошук між елементами бази даних та фільтрувати їх за певними характеристиками. В ролі бази даних, використовується NoSQL база даних під назвою MongoDB. Вона не має таблиць, як SQL бази даних, а натомість зберігає дані у колекціях формату JSON. Цей формат надзвичайно зручний для налагодження клієнт-серверної взаємодії.

Працюючи з великою кількістю особистих даних, важко не упустити важливу інформацію. Розроблена програма дозволяє будь-якому читачеві тримати найважливіше про книги в одному місці, а саме: книги, їх характеристики, статус (прочитано, в процесі, покинуто, в планах) і цитати, кожна з яких обов'язково має прив'язку до конкретної книги, доданої юзером.

Використання підходів об'єктно-орієнтованого програмування дозволяє організувати код краще, спрощує реалізацію функціоналу, дозволяє побудувати прозору програмну архітектуру, що полегшує процес дорозробки програми в майбутньому з мінімальним втручанням у вже написаний програмний код.

Метою роботи є написання програми з можливістю створювати, редагувати та видаляти книги та цитати певного користувача, який заздалегідь зареєструвався на даному веб-ресурсі. Результатом роботи має стати закріплення вивченого з дисципліни "Об'єктно-орієнтоване програмування", а також практика розробки Backend-логіки на мові C# і Frontend на JS-фреймворці React.

СПЕЦИФІКАЦІЯ ПРОГРАМИ

1.1 Функції програми

Додаток надаватиме користувачу наступні можливості:

- реєстрація (створення нового облікового запису);
- авторизація до вже створеного облікового запису;
- додавання книг до списку обраних книг юзера, який поточно залогінений на ресурсі;
- видалення книг зі списку обраних книг;
- додавання цитат до списку всіх цитат користувача, який поточно залогінений на ресурсі;
- видалення цитат зі списку всіх цитат;
- пошук книг за певними критеріями: автор, статус, назва;
- перегляд списку всіх книг, доданих користувачем
- перегляд списку всіх цитат, доданих користувачем
- перегляд окремої сторінки для конкретної книги, де відображена вся інформація по цій книзі.

Усі дані користувачів: паролі, особисті дані (ім'я, прізвище, адрес електронної пошти), додані книги, додані цитати будуть зберігатися у NoSQL базі даних MongoDB, яка запущена локально. Фільтрація, додавання,

видалення певних даних миттєво призводить до зміни інтерфейсу користувача.

1.2 Інтерфейс користувача

Після запуску додатку вперше можна буде побачити сторінку привітання. Користувачу доступна тільки одна вкладка у боковій панелі - “Home”, а також показуються кнопки для реєстрації та входу до облікового запису в шапці сайту (див. рис. 1.1).

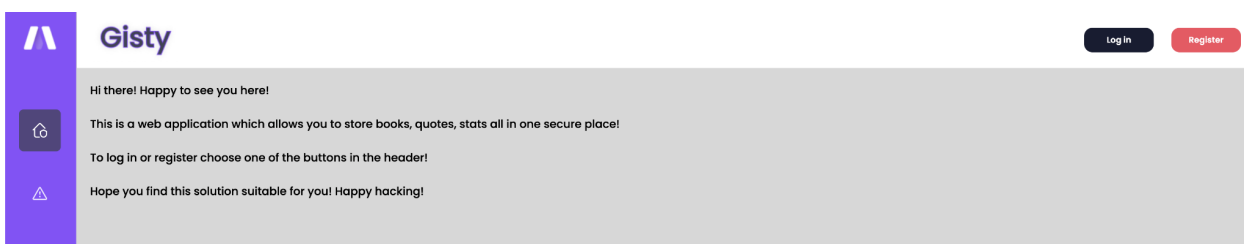
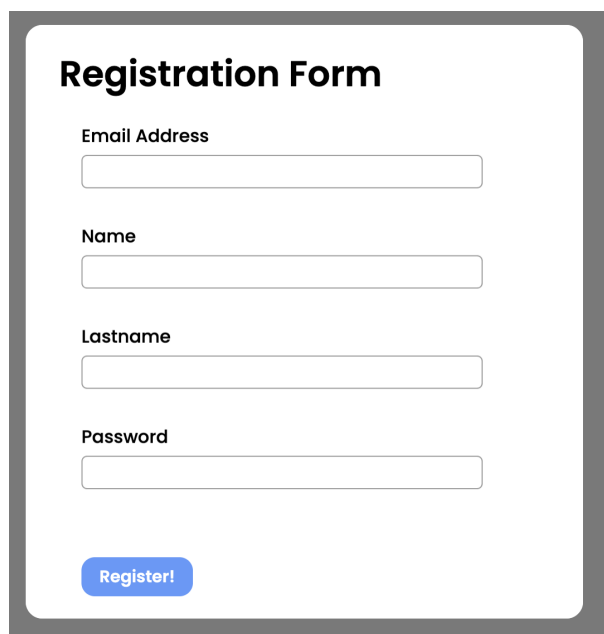


Рисунок 1.1 – Сторінка привітання

Для створення нового облікового запису юзер має натиснути на кнопку “Register” у правому верхньому куті графічного інтерфейсу. В такому разі користувач побачить модальне вікно, яке запросить деякі дані про користувача, проведе їх валідацію і після кнопки “Register” зареєструє новий запис у базі даних.



The image shows a registration form titled "Registration Form". It contains four input fields: "Email Address", "Name", "Lastname", and "Password". Below the fields is a blue button labeled "Register!". The form is enclosed in a white box with a grey border.

Рисунок 1.2 – Вікно реєстрації користувача

Після успішного заповнення форми реєстрації, або аналогічно форми для входу до облікового запису в разі, якщо користувач вже зареєстрований, користувач побачить ще дві опції в боковій панелі для переходу на сторінку списку цитат і для відкриття вікна для додавання нової книги, а також побачить список книг. Якщо це відбувається відразу після реєстрації, цей список, як і список цитат, буде пустим.

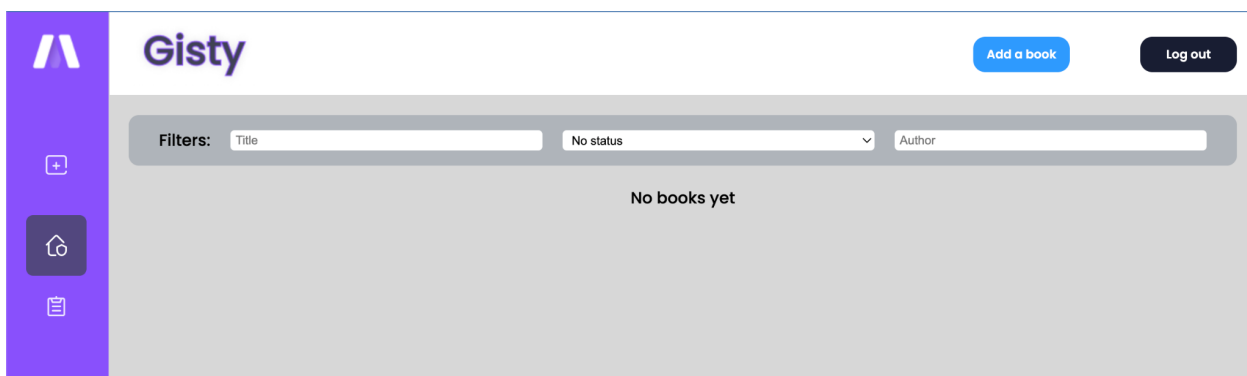


Рисунок 1.3 – Порожній список книг після реєстрації

Натиснувши на третю за рахунком опцію в боковому меню, користувач побачить іншу сторінку - список доданих цитат.

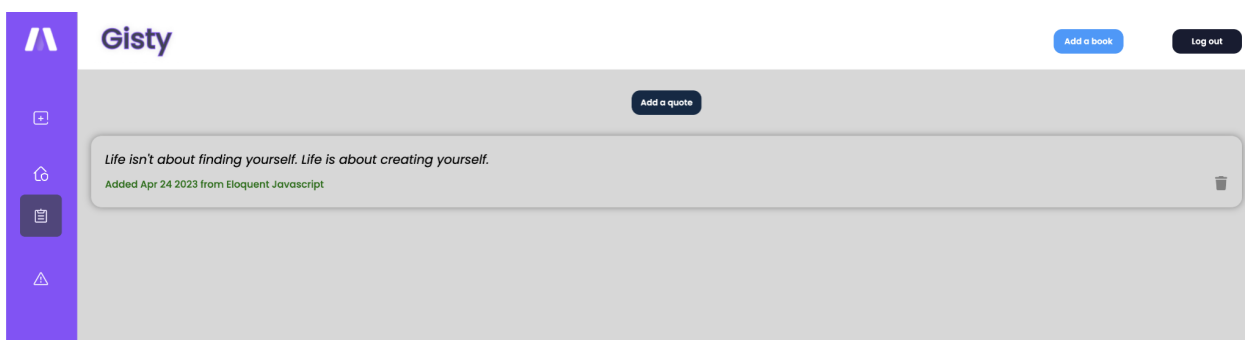


Рисунок 1.4 – Список цитат

Для зручного користування та знаходження необхідної книги серед усіх доданих, на сайті реалізована фільтрація книг за автором, статусом та назвою. Вся інформація буде зберігатися та оновлюватися миттєво після взаємодії з інтерфейсом (див. рис. 1.5).

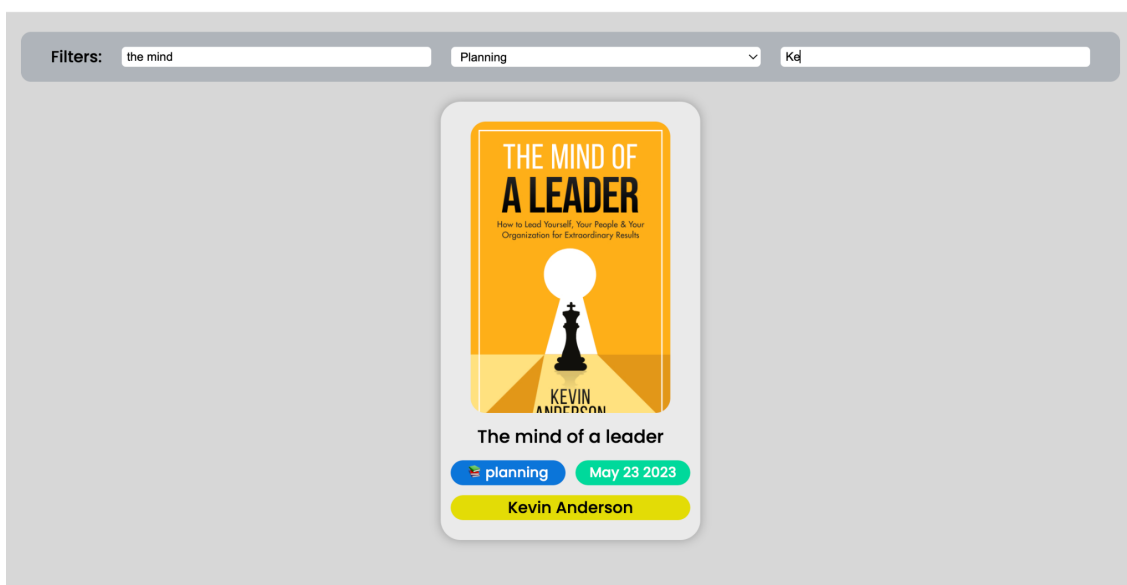


Рисунок 1.5 – Відображення відфільтрованих книг

Для видалення книги з переліку необхідно перейти на індивідуальну сторінку книги і натиснути кнопку “Delete a book”, після чого відкриється

вікно для підтвердження операції. Також буде поле типу “checkbox” для того, щоб контролювати, чи видаляти прив’язані цитати до книги, яку видаляємо.

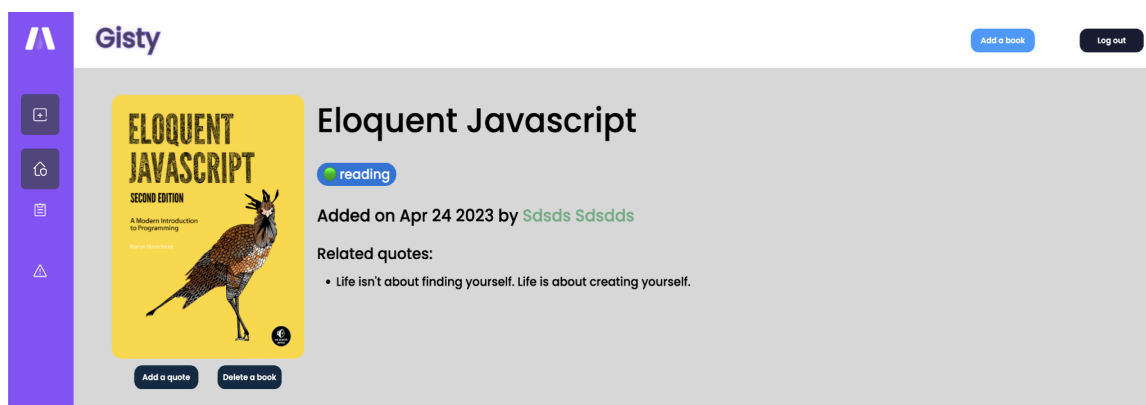


Рисунок 1.6 – Індивідуальна сторінка книги

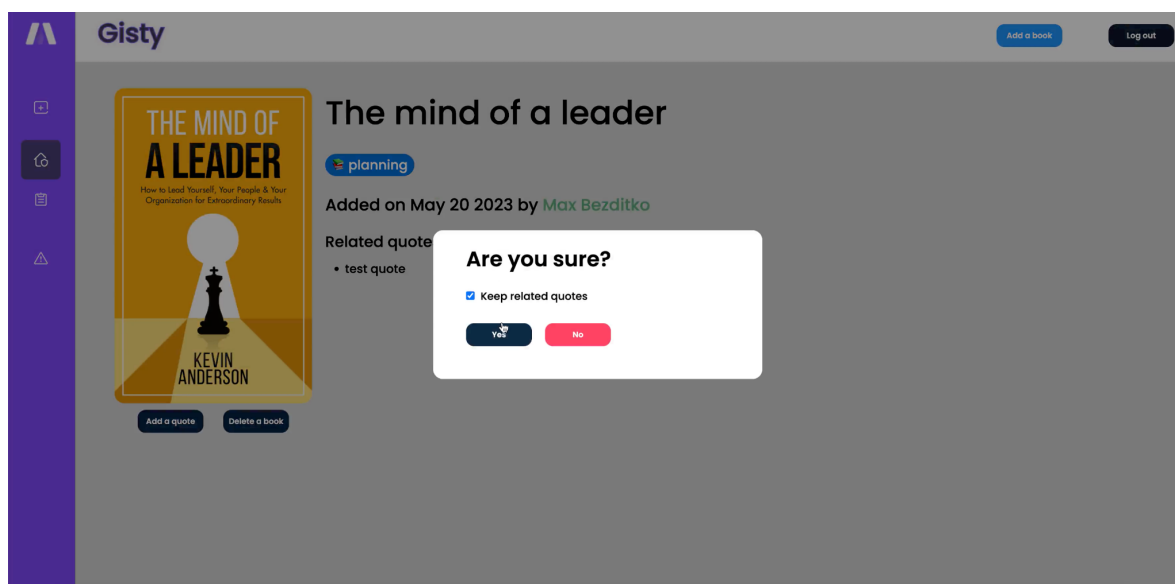
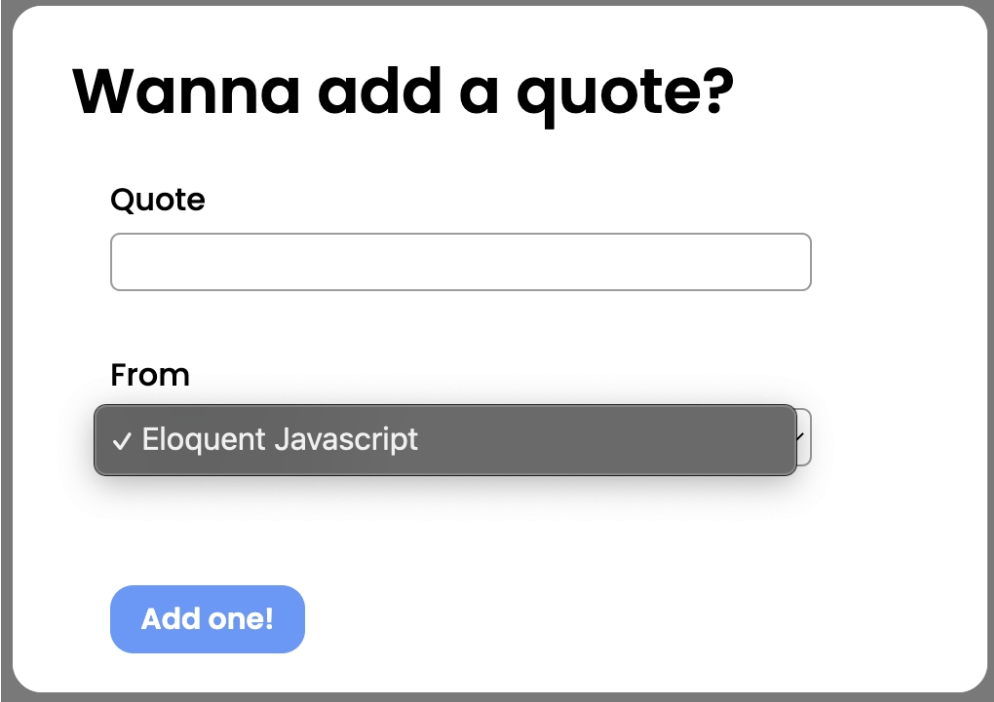


Рисунок 1.7 – Вікно підтвердження видалення книги

Для додавання цитати необхідно перейти на сторінку, де відображений список доданих цитат. Потім натиснути кнопку “Add a quote”, заповнити форму, де необхідно вказати зміст цитати, а також книгу, до якої прив’язана цитата.



Wanna add a quote?

Quote

From

✓ Eloquent Javascript

Add one!

Рисунок 1.8 – Вікно додавання цитати

2 ПРОЕКТНА СПЕЦИФІКАЦІЯ

2.1 Об'єктна модель програми

Розроблена програма являє собою набір класів, які в свою чергу поділяються на моделі (див. рис. 2.1), контролери (див. рис. 2.2) і сервіси (див. рис. 2.3).

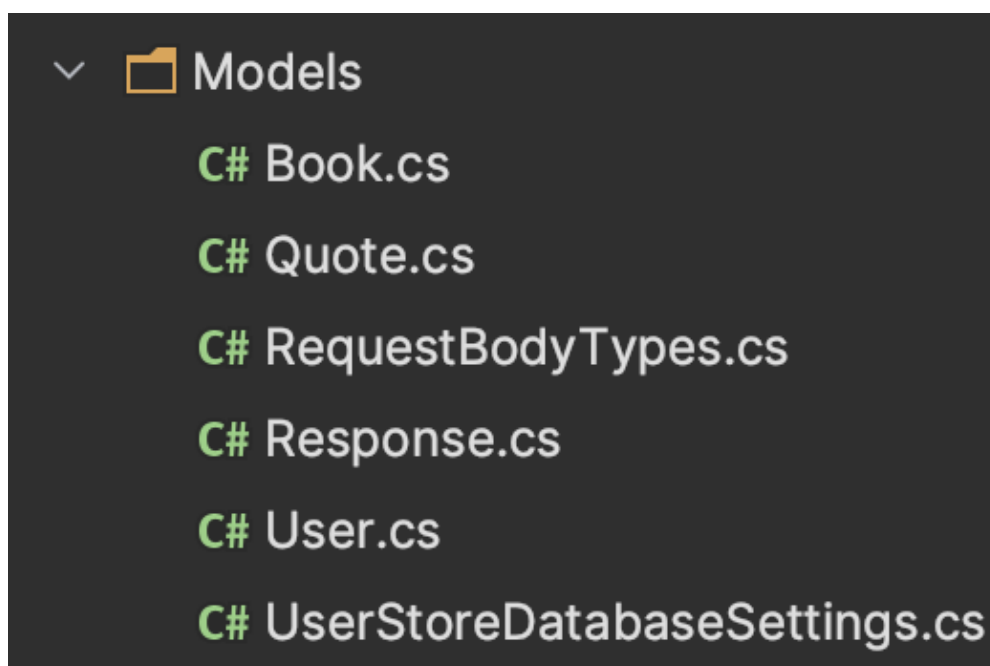


Рисунок 2.1 – Класи-моделі

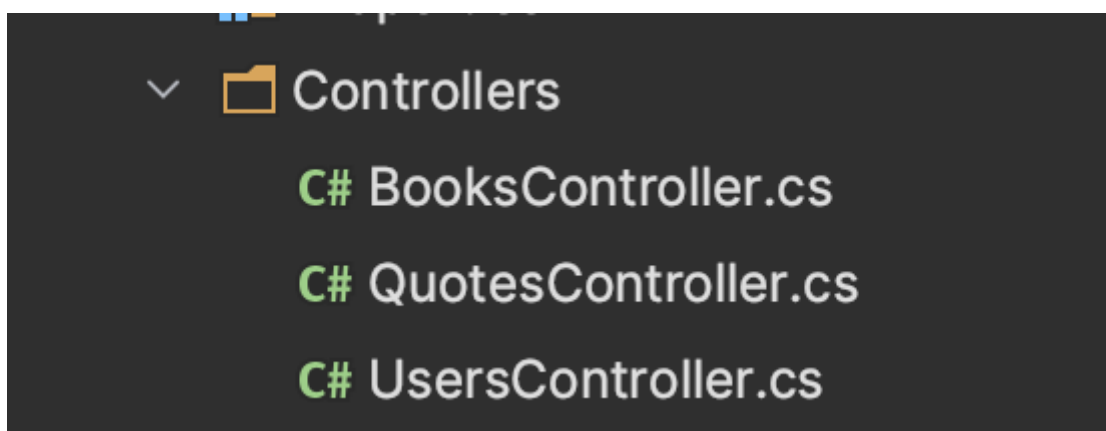


Рисунок 2.2 – Класи-контролери

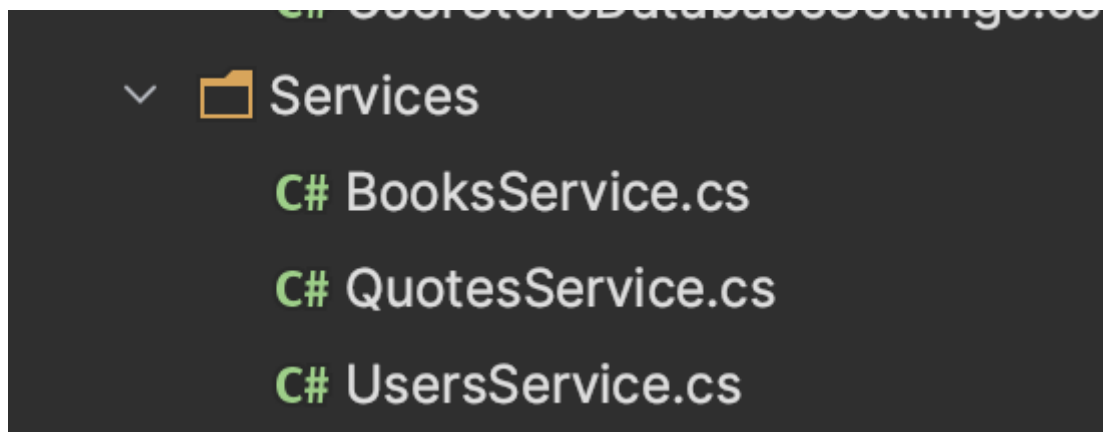


Рисунок 2.3 – Класи-сервіси

Класи-контролери успадковуються від класу `ControllerBase`, який є частиною фреймворку `Asp .Net Core`. У контролерах визначаються ендпойнти, які здатні обробляти HTTP-запити, викликаючи потрібні функції відповідних класів-сервісів, повертати дані користувачеві.

Класи-сервіси використовуються для імплементації логіки взаємодії з базою даних, фільтрації контенту.

2.2 Реалізація функцій програми

Найголовніший клас, з якого починається робота програми, та де проходить основна конфігурація проекту, - це клас Program. У ньому реєструються контролери та сервіси, конфігурується додаток та запускається додаток.

```
builder.Services.Configure<UserStoreDatabaseSettings>(  
    builder.Configuration.GetSection(key: "UsersDatabase"));  
  
builder.Services.AddSingleton<UsersService>();  
builder.Services.AddSingleton<BooksService>();  
builder.Services.AddSingleton<QuotesService>();  
  
builder.Services // IServiceCollection  
    .AddControllers()  
    .AddJsonOptions(  
        options => options.JsonSerializerOptions.PropertyNamingPolicy = null);  
  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();|  
  
var app :WebApplication = builder.Build();  
  
app.UseCors(builder :CorsPolicyBuilder => builder.AllowAnyOrigin());
```

Рисунок 2.4 – Реєстрація сервісів та контролерів

Для обробки різних посилань до REST API, реалізовано три контролера, окремо для цитат, користувачів та книг. Розглянемо приклад обробки запиту для реєстрації користувача:

```
[HttpPost( template: "Register")]
Maksym Bezditko
public async Task<IActionResult> Post(User newUser)
{
    await _userService.CreateAsync(newUser);

    return Ok(newUser);
}
```

Рисунок 2.5 – Реєстрація сервісів та контролерів

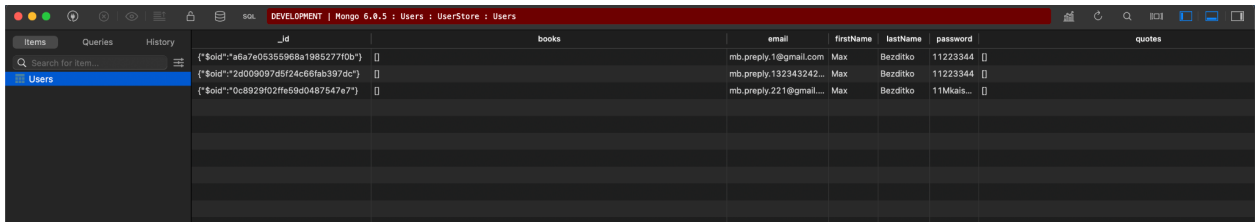
Реалізація методу CreateAsync сервісу користувачів:

```
1 usage Maksym Bezditko
public async Task CreateAsync(User newBook) =>
    await _booksCollection.InsertOneAsync(newBook);
```

Рисунок 2.6 – Редагування бази даних

2.3 Формат даних

Додаток використовує NoSQL базу даних під назвою MongoDB. База даних не має традиційних таблиць, як SQL бази даних. Натомість зберігає дані у виді JSON-колекцій. Взаємодія з базою даних MongoDB через фреймворк Asp .Net Core відбувається за допомогою залежності MongoDB.Driver. Дані зберігаються в форматі JSON-колекцій (див. рис. 2.7).



_id	books	email	firstName	lastName	password	quotes
["\$oid":"a6a7e05355968a1985277f0b"]		mb.preply.1@gmail.com	Max	Bezditko	11223344	
["\$oid":"2d009097d5f24c66fab397dc"]		mb.preply.132343242...	Max	Bezditko	11223344	
["\$oid":"0c8929f02ffe69d0487547e7"]		mb.preply.221@gmail....	Max	Bezditko	11Mkals...	

Рисунок 2.7 – Ілюстрація записів у базі даних

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Установка програми

Для того щоб запустити програму локально у себе на комп'ютері, вам необхідно мати встановленими NodeJS, NPM, Mongo, mongosh, .Net SDK на вашому ПК. Установка додатку відбувається наступним чином, ви маєте дві папки в корні проекту: frontend та backend. Спочатку перейдіть у папку backend, відкрийте проект у Visual Studio чи JetBrains Rider, натисніть кнопку “Start” для запуску серверу. Сервер запуститься локально на порту 5032 за замовчуванням. При запуску сервера обирайте http-протокол передачі даних. Далі перейдіть в папку frontend з корню проекту, відкрийте там термінал та пропишіть команду “npm i && npm start”. Дочекайтеся запуску додатку. Нова вкладка автоматично відкриється у вікні браузера. За замовчуванням, клієнтна частина запуститься локально на порті 3000.

3.2 Робота з програмою

Сторінка привітання зображена на рисунку 3.1.

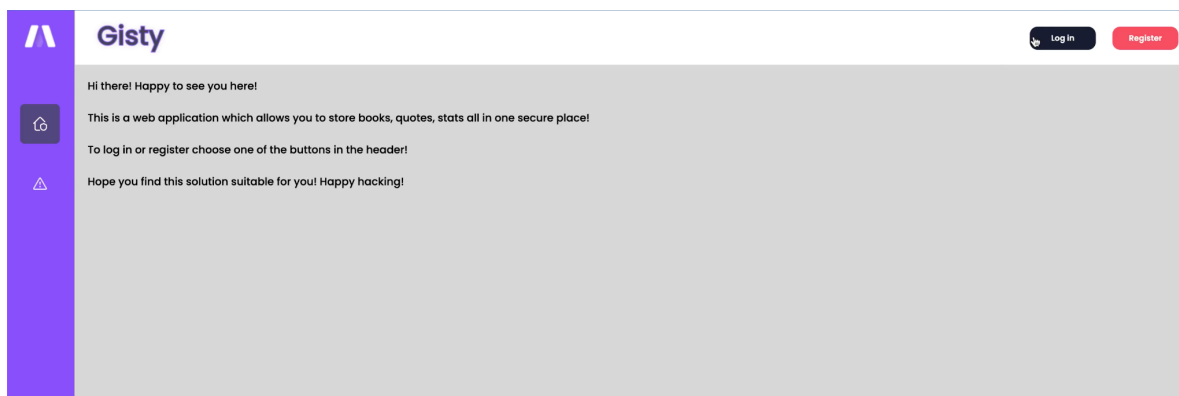
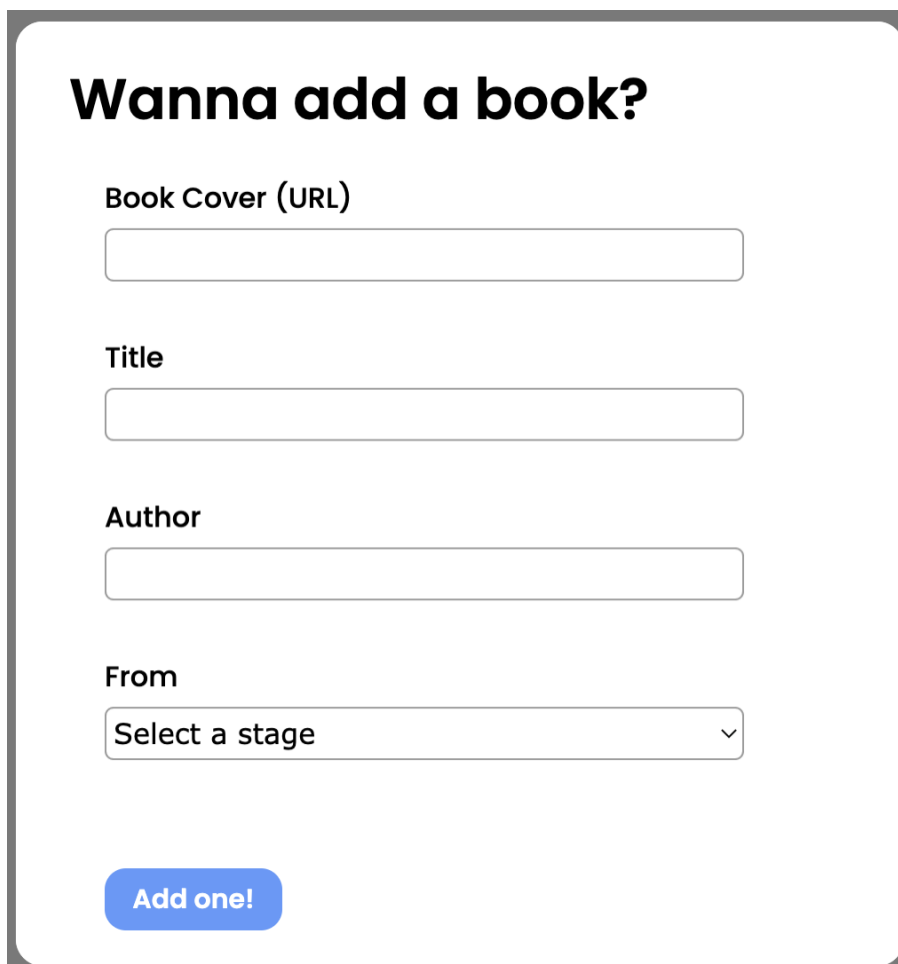


Рисунок 3.1 – Сторінка привітання

Перелік книг, цитат та функціонал для опрацювання даних доступні після реєстрації чи аутентифікації.

Для додавання книги необхідно натиснути на найпершу іконку і боковій панелі або на кнопку “Add a book” в шапці сайту. Після цього відкриється вікно, де потрібно буде ввести дані про книгу.



Wanna add a book?

Book Cover (URL)

Title

Author

From

Select a stage ▾

Add one!

Рисунок 3.2 – Додавання нової книги

Після натискання на кнопку “Add one!”, нова книга з’явиться у загальному переліку доданих книг. За бажанням, можна перейти на сторінку окремої книги, натиснувши на неї. Відкриється нова сторінка присвячена одній конкретній книзі, яку обрав користувач

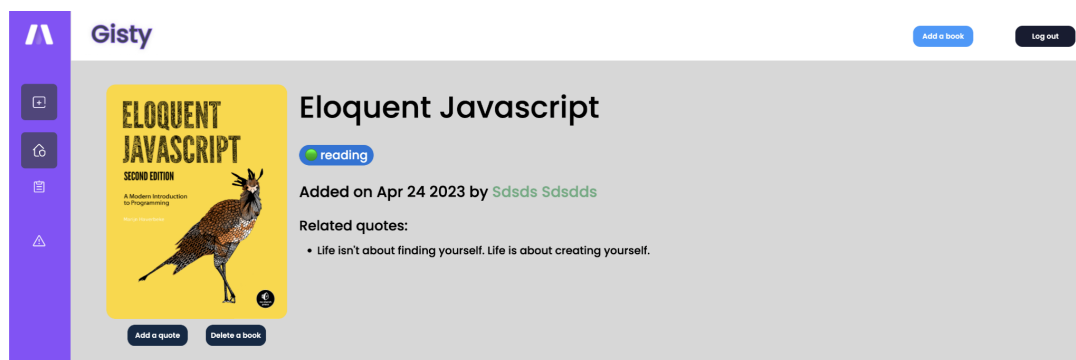


Рисунок 3.3 – Відкрита сторінка окремої книги

Для видалення книги потрібно натиснути на кнопку “Delete a book” та у вікні, яке відкриється підтвердити операцію.

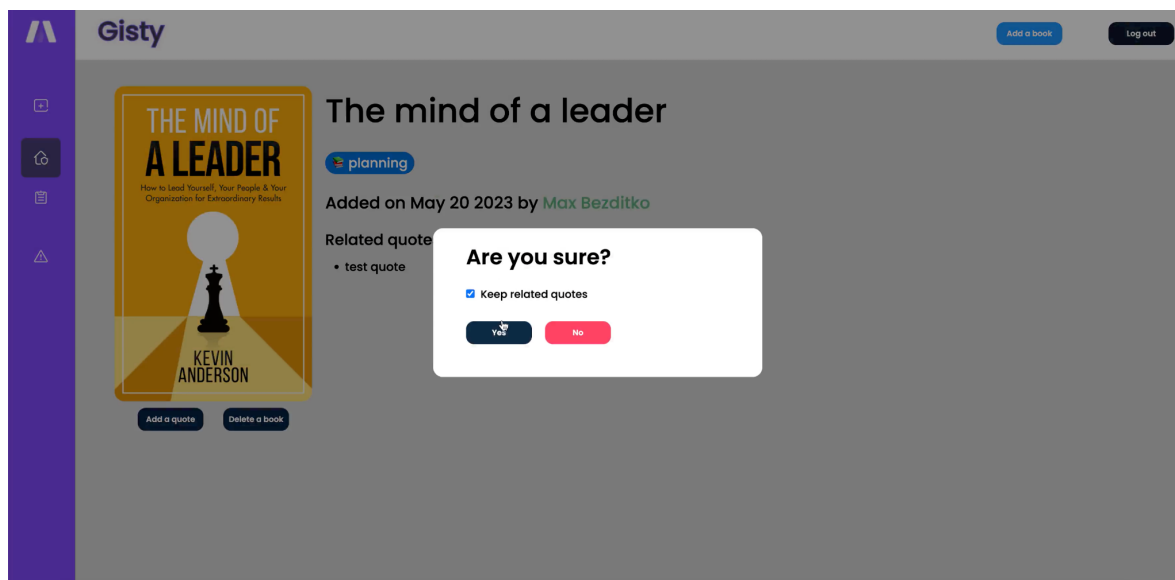


Рисунок 3.4 – Видалення книги

Користувач може натиснути на третю за рахунком іконку в боковій панелі, що відкриє перелік всіх доданих цитат користувача.

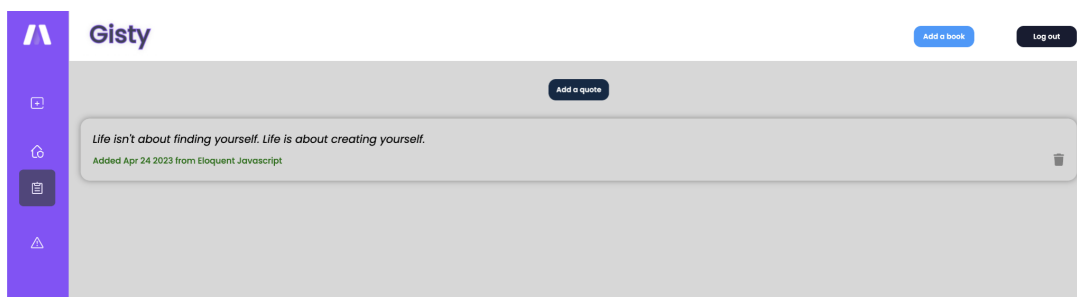


Рисунок 3.5 – Перелік цитат

Користувач має змогу натиснути іконку корзини на одній з цитат, це видалить обрану цитату з загального переліку.

Для додавання цитати потрібно натиснути кнопку “Add a quote”, та заповнити дані про цитату, яку хочете додати.

The form is titled "Wanna add a quote?". It contains two input fields: "Quote" and "From". The "From" field has a dropdown menu with "Eloquent Javascript" selected. At the bottom is a blue button labeled "Add one!".

Рисунок 3.6 – Додавання цитати

Після натискання на кнопку “Add one!”, нова цитати з’явиться у загальному переліку доданих цитат.

Для фільтрації книг необхідно перейти на сторінку відображення загального переліку книг і почати вводити дані у компоненті “Filters”, після чого фільтрація відбувається автоматично.

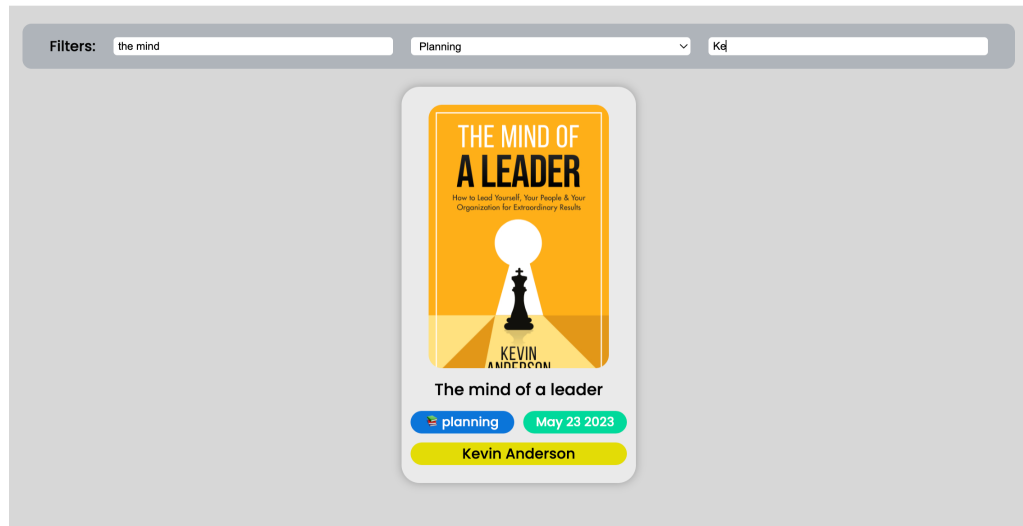


Рисунок 3.6 – Фільтрація книг

Для завершення роботи програми необхідно вийти з облікового запису, натиснувши кнопку “Log out” у правій верхній частині екрану. Після цього юзер знову побачить сторінку привітання.

ВИСНОВКИ

Під час виконання курсової роботи була розроблена програма «Особиста бібліотека», яка дозволяє вести облік книг, які читає користувач та цитат, які той додав під час прочитання тієї чи іншої книги. Користувач має можливість видаляти, редагувати, додавати дані.

У результаті виконання курсової роботи було отримано додаток, який здатний значно покращити та оптимізувати спосіб відслідковування релевантної літератури для читача.

Користуватися додатком можна з будь-яких електронних пристроїв, дані користувача не прив'язуються до конкретного браузера чи девайсу, а зберігаються в базі даних, тобто для їх отримання потрібно знати тільки адресу електронної пошти користувача та пароль від облікового запису.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бондарєв В. М. Основи програмування на JavaScript : навчальний посібник / В. М. Бондарєв. – Харків : Коряк С. Ф., 2015. – 182 с.
2. С# / BestProg. Програмування: теорія та практика. URL: https://www.bestprog.net/uk/sitemap_ua/c-3/ (дата звернення: 25.05.2022)
3. ДСТУ 3008:2015 Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. К.: ДП «УкрНДНЦ», 2016. 26 с.
4. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. К.: ДП «УкрНДНЦ», 2016. – 16 с.
5. Документація платформи ASP .NET Core. URL: <https://learn.microsoft.com/en-us/dotnet/> (дата звернення 25.05.2022).
6. Документація мови програмування JavaScript. URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата звернення 25.05.2022).
7. Документація бібліотеки React. URL: <https://ru.reactjs.org/> (дата звернення 25.05.2022).

ДОДАТОК А

Код програми

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<UserStoreDatabaseSettings>(
    builder.Configuration.GetSection("UsersDatabase"));

builder.Services.AddSingleton<UsersService>();
builder.Services.AddSingleton<BooksService>();
builder.Services.AddSingleton<QuotesService>();

builder.Services
    .AddControllers()
    .AddJsonOptions(
        options =>
options.JsonSerializerOptions.PropertyNamingPolicy = null);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseCors(build => build.AllowAnyOrigin());

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
```

```
        app.UseSwaggerUI();
    }

    app.UseHttpsRedirection();

    app.UseAuthorization();

    app.MapControllers();

    app.Run();
    using backend.Models;
    using backend.Services;
    using Microsoft.AspNetCore.Mvc;

    namespace backend.Controllers;

    [ApiController]
    [Route("api/[controller]")]

    public class BooksController : ControllerBase
    {
        private readonly BooksService _booksService;
        private readonly UsersService _usersService;

        public BooksController(BooksService booksService,
            UsersService usersService)
        {
```

```

        _booksService = booksService;
        _usersService = usersService;
    }

    [HttpGet("Get/{userId:length(24)}")]
    public async Task<IActionResult> GetAll(string userId)
    {
        var user = await _usersService.GetAsync(userId);

        if (user is null)
            return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });

        return Ok(_booksService.GetAsync(userId));
    }

    [HttpPost("Add")]
    public async Task<IActionResult> Add([FromBody]
AddBookRequestBody body)
    {
        var user = await _usersService.GetAsync(body.userId);

        if (user is null)
            return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });

        await _booksService.AddBookAsync(body.userId, body.book);

        return Ok();
    }

```

```

    }

    [HttpPost("Delete")]
    public async Task<IActionResult> Delete([FromBody]
DeleteBookRequestBody body)
    {
        var user = await _userService.GetAsync(body.userId);

        if (user is null)
            return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });

        await _booksService.DeleteBookAsync(body.userId,
body.bookId);

        return Ok();
    }
}

using backend.Models;
using backend.Services;
using Microsoft.AspNetCore.Mvc;

namespace backend.Controllers;

[ApiController]
[Route("api/[controller]")]

public class QuotesController : ControllerBase
{

```

```

private readonly QuotesService _quotesService;
private readonly UsersService _usersService;

public QuotesController(QuotesService quotesService,
UsersService usersService)
{
    _quotesService = quotesService;
    _usersService = usersService;
}

[HttpGet("Get/{userId:length(24)}")]
public async Task<IActionResult> GetAll(string userId)
{
    var user = await _usersService.GetAsync(userId);

    if (user is null)
        return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });

    return Ok(_quotesService.GetAsync(userId));
}

[HttpPost("Add")]
public async Task<IActionResult> Add([FromBody]
AddQuoteRequestBody body)
{
    var user = await _usersService.GetAsync(body.userId);

    if (user is null)

```

```
        return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });
```

```
        await _quotesService.AddQuoteAsync(body.userId,
body.quote);
```

```
        return Ok();
    }
```

```
[HttpPost("Delete")]
public async Task<IActionResult> Delete([FromBody]
DeleteQuoteRequestBody body)
{
    var user = await _userService.GetAsync(body.userId);

    if (user is null)
        return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });
```

```
        await _quotesService.DeleteQuoteAsync(body.userId,
body.quoteId);
```

```
        return Ok();
    }
```

```
[HttpPost("DeleteAssociated")]
public async Task<IActionResult> DeleteAssociated([FromBody]
DeleteAssociatedQuotesRequestBody body)
{
    var user = await _userService.GetAsync(body.userId);
```

```

        if (user is null)
            return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });

        await
_quotesService.DeleteQuotesForBookAsync(body.userId,
body.bookId);

        return Ok();
    }
}
using backend.Models;
using backend.Services;
using Microsoft.AspNetCore.Mvc;

namespace backend.Controllers;

[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase
{
    private readonly UsersService _userService;

    public UsersController(UsersService userService) =>
        _userService = userService;

    [HttpGet("Get")]

```

```

public async Task<List<User>> Get() =>
    await _userService.GetAsync();

[HttpGet("Get/{userId:length(24)}")]
public async Task<ActionResult<User>> Get(string userId)
{
    var user = await _userService.GetAsync(userId);

    if (user is null)
    {
        return NotFound(new Response { Status = 404, Message
= "User not found!", Succeeded = false });
    }

    return user;
}

[HttpPost("CheckExistence")]
public async Task<IActionResult> Exists([FromBody]
CheckExistenceRequestBody credentials)
{
    var users = await _userService.GetAsync();

    var found = users.Any(b => b.email == credentials.email
&& b.password == credentials.password);

    if (!found) return NotFound(new Response { Status = 404,
Message = "User not found!", Succeeded = false });
    {
        var userId = users.Find(b => b.email ==
credentials.email && b.password == credentials.password)!.id;

```



```

        return Ok(new Response { Status = 200, Message =
userId, Succeeded = true });
    }

```

```

    }

```

```

[HttpPost("EmailExistence")]
public async Task<IActionResult> EmailExists([FromBody]
CheckEmailRequestBody credentials)
{
    var users = await _userService.GetAsync();

    var found = users.Any(b => b.email == credentials.email);

    if (found) return Ok(new Response { Succeeded = false,
Message = "found", Status = 200 });
    {
        return Ok(new Response{ Succeeded = true, Message =
"not found", Status = 200 });
    }
}

```

```

[HttpPost("Register")]
public async Task<IActionResult> Post(User newUser)
{
    await _userService.CreateAsync(newUser);

    return Ok(newUser);
}

```

```

[HttpPost("Delete/{userId:length(24)}")]
public async Task<IActionResult> Delete(string userId)
{
    var user = await _userService.GetAsync(userId);

    if (user is null)
    {
        return NotFound();
    }

    await _userService.RemoveAsync(userId);

    return NoContent();
}
}
namespace backend.Models;

public class Book
{
    public string id { get; set; } = null!;

    public string author { get; set; } = null!;

    public string title { get; set; } = null!;

    public string coverUrl { get; set; } = null!;

    public string status { get; set; } = null!;

    public string date { get; set; } = null;
}

```

```
}  
namespace backend.Models;  
  
public class Quote  
{  
    public string id { get; set; } = null!;  
  
    public string content { get; set; } = null!;  
  
    public string associatedWithBookId { get; set; } = null!;  
  
    public string date { get; set; } = null!;  
}  
namespace backend.Models;  
  
public class CheckExistenceRequestBody  
{  
    public string email { get; set; } = null!;  
    public string password { get; set; } = null!;  
}  
  
public class CheckEmailRequestBody  
{  
    public string email { get; set; } = null!;  
}  
  
public class GetBooksRequestBody  
{  
    public string userId { get; set; } = null!;  
}
```

```
public class DeleteBookRequestBody
{
    public string userId { get; set; }
    public string bookId { get; set; }
}
```

```
public class DeleteAssociatedQuotesRequestBody
{
    public string userId { get; set; }
    public string bookId { get; set; }
}
```

```
public class DeleteQuoteRequestBody
{
    public string userId { get; set; }
    public string quoteId { get; set; }
}
```

```
public class AddBookRequestBody
{
    public string userId { get; set; }
    public Book book { get; set; }
}
```

```
public class AddQuoteRequestBody
{
    public string userId { get; set; }
    public Quote quote { get; set; }
}
```

```
namespace backend.Models;
```

```
public class Response
{
    public bool Succeeded { get; set; }

    public int Status { get; set; }

    public string? Message { get; set; }
}
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
```

```
namespace backend.Models;
```

```
public class User
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string id { get; set; }

    public string firstName { get; set; } = null!;

    public string lastName { get; set; } = null!;

    public string email { get; set; } = null!;

    public string password { get; set; } = null!;

    public List<Book> books { get; set; } = new ();
```

```

        public List<Quote> quotes { get; set; } = new ();
    }
namespace backend.Models;

public class UserStoreDatabaseSettings
{
    public string ConnectionString { get; set; } = null!;

    public string DatabaseName { get; set; } = null!;

    public string UsersCollectionName { get; set; } = null!;
}
using backend.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace backend.Services;

public class BooksService
{
    private readonly IMongoCollection<User> _usersCollection;

    public BooksService(
        IOptions<UserStoreDatabaseSettings>
userStoreDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            userStoreDatabaseSettings.Value.ConnectionString);
    }

```

```

var mongoDatabase = mongoClient.GetDatabase(
    userStoreDatabaseSettings.Value.DatabaseName);

_usersCollection = mongoDatabase.GetCollection<User>(
    userStoreDatabaseSettings.Value.UsersCollectionName);
}

public List<Book> GetAsync(string userId)
{
    var userFilter = Builders<User>.Filter.Eq(u => u.id,
userId);
    var user =
_usersCollection.Find(userFilter).FirstOrDefault();

    return user.books;
}

public async Task AddBookAsync(string userId, Book newBook)
{
    var filter = Builders<User>.Filter.Eq(u => u.id, userId);
    var update = Builders<User>.Update.Push(u => u.books,
newBook);

    await _usersCollection.UpdateOneAsync(filter, update);
}

public async Task DeleteBookAsync(string userId, string
bookId)
{

```

```

        var userFilter = Builders<User>.Filter.Eq(u => u.id,
userId);

        var userUpdate = Builders<User>.Update.PullFilter(u =>
u.books, b => b.id == bookId);

        await _usersCollection.UpdateOneAsync(userFilter,
userUpdate);
    }
}

using backend.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace backend.Services;

public class QuotesService
{
    private readonly IMongoCollection<User> _usersCollection;

    public QuotesService(
        IOptions<UserStoreDatabaseSettings>
userStoreDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            userStoreDatabaseSettings.Value.ConnectionString);

        var mongoDatabase = mongoClient.GetDatabase(
            userStoreDatabaseSettings.Value.DatabaseName);

        _usersCollection = mongoDatabase.GetCollection<User>(

```



```

        userStoreDatabaseSettings.Value.UsersCollectionName);
    }

    public List<Quote> GetAsync(string userId)
    {
        var userFilter = Builders<User>.Filter.Eq(u => u.id,
userId);
        var user =
_usersCollection.Find(userFilter).FirstOrDefault();

        return user.quotes;
    }

    public async Task AddQuoteAsync(string userId, Quote
newQuote)
    {
        var filter = Builders<User>.Filter.Eq(u => u.id, userId);
        var update = Builders<User>.Update.Push(u => u.quotes,
newQuote);

        await _usersCollection.UpdateOneAsync(filter, update);
    }

    public async Task DeleteQuoteAsync(string userId, string
quoteId)
    {
        var userFilter = Builders<User>.Filter.Eq(u => u.id,
userId);
        var userUpdate = Builders<User>.Update.PullFilter(u =>
u.quotes, b => b.id == quoteId);

        await _usersCollection.UpdateOneAsync(userFilter,
userUpdate);
    }

```

```

    }

    public async Task DeleteQuotesForBookAsync(string userId,
string bookId)
    {
        var userFilter = Builders<User>.Filter.Eq(u => u.id,
userId);

        var userUpdate = Builders<User>.Update.PullFilter(u =>
u.quotes, b => b.associatedWithBookId == bookId);

        await _usersCollection.UpdateOneAsync(userFilter,
userUpdate);
    }
}

using backend.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace backend.Services;

public class UsersService
{
    private readonly IMongoCollection<User> _booksCollection;

    public UsersService(
        IOptions<UserStoreDatabaseSettings>
userStoreDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            userStoreDatabaseSettings.Value.ConnectionString);
    }
}

```

```
var mongoDatabase = mongoClient.GetDatabase(  
    userStoreDatabaseSettings.Value.DatabaseName);  
  
_booksCollection = mongoDatabase.GetCollection<User>(  
    userStoreDatabaseSettings.Value.UsersCollectionName);  
}  
  
public async Task<List<User>> GetAsync() =>  
    await _booksCollection.Find(_ => true).ToListAsync();  
  
public async Task<User?> GetAsync(string id) =>  
    await _booksCollection.Find(x => x.id ==  
id).FirstOrDefaultAsync();  
  
public async Task CreateAsync(User newBook) =>  
    await _booksCollection.InsertOneAsync(newBook);  
  
public async Task RemoveAsync(string id) =>  
    await _booksCollection.DeleteOneAsync(x => x.id == id);  
}
```