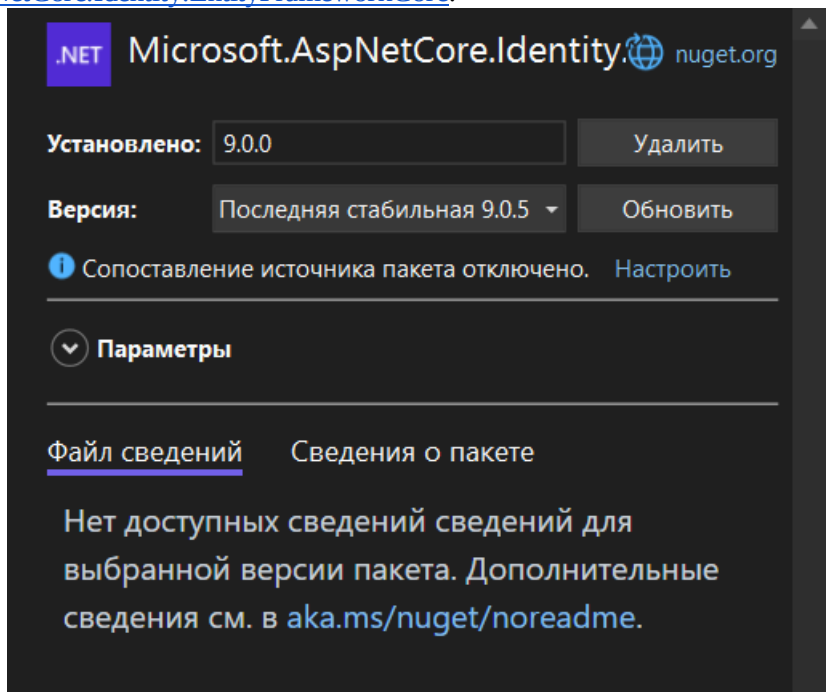
Лабораторна робота №5

Реалізація спроможностей реєстрації, аутентифікації та авторизації засобами фреймворку ASP.NET Core Identity.

Виконав студент групи 302-ТК

Писаренко Дмитрій

1. Додати в проєкт {Назва тематики}.Infrastructure нову залежність на NuGET пакет Microsoft.AspNetCore.Identity.EntityFrameworkCore.

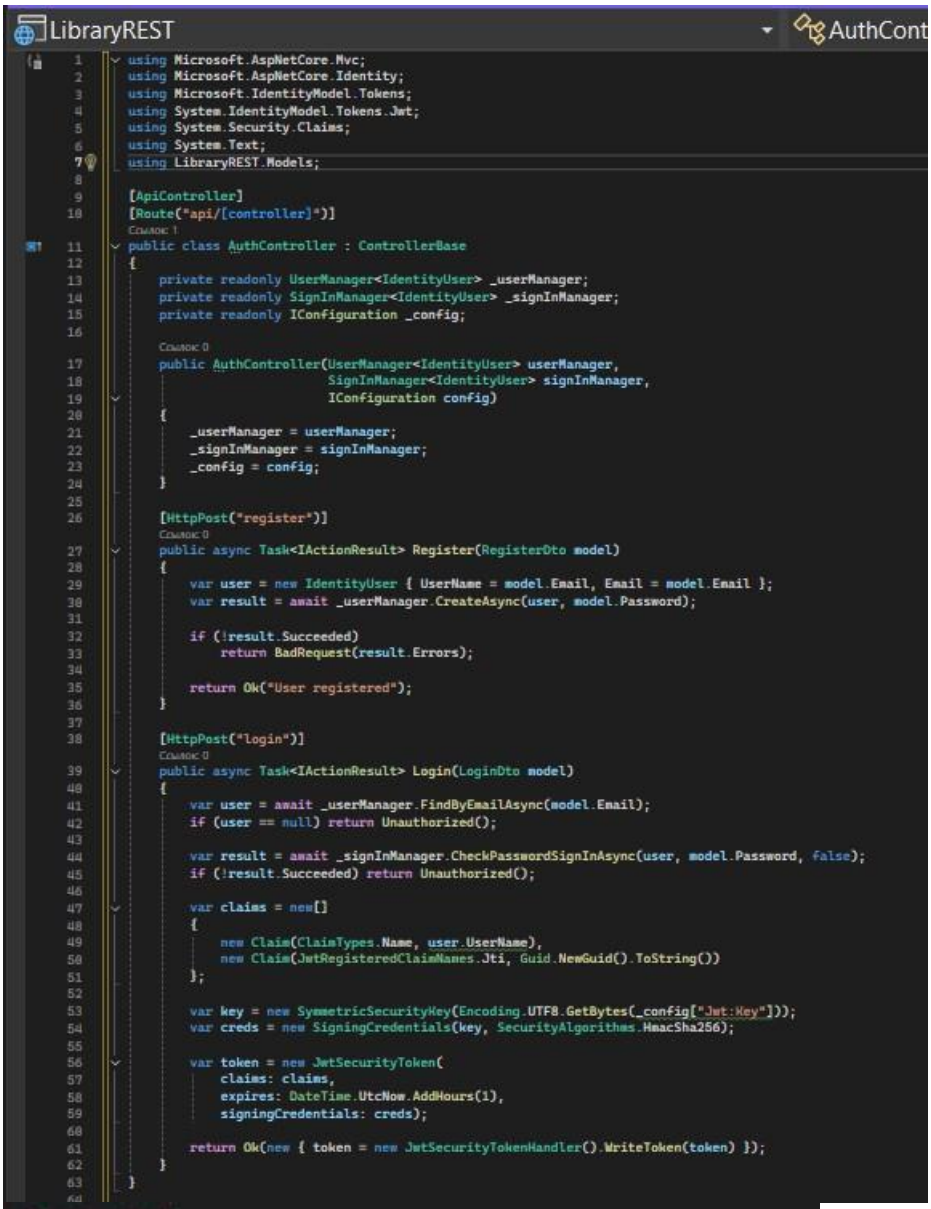


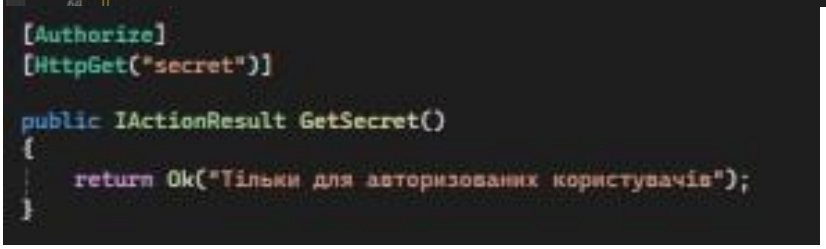
2. Модифікуйте створену вами у попередніх лабораторних роботах модель (набір класів), додавши до неї сутність користувача, яка буде наслідуватися від класу IdentityUser із фреймворку ASP.NET Core Identity. Модифікуйте, створений у третій лабораторній, DbContext таким чином, щоб він наслідувався від класу IdentityDbContext<T>, де узагальнюючим типом буде створена вами сутність користувача.

```
namespace Library.Infrastructure.Models
{
    Ссылок: 2
    public class AppUser : IdentityUser
    {
        // Додаткові властивості користувача (опціонально)
        Ссылок: 0
        public string FullName { get; set; }
    }
}
```

```
namespace Library.Infrastructure.Data
{

    public class LibraryDbContext : IdentityDbContext<AppUser>
    {
```

3. Модифікуйте проєкт {Назва тематики}.REST, щоб він виставляв на зовні вбудовані у ASP.NET Core Identity кінцеві точки для реєстрації та аутентифікації, при цьому аутентифікація повинна створювати Bearer або JWT токени. Можете спиратися на офіційний туторіал від Microsoft.

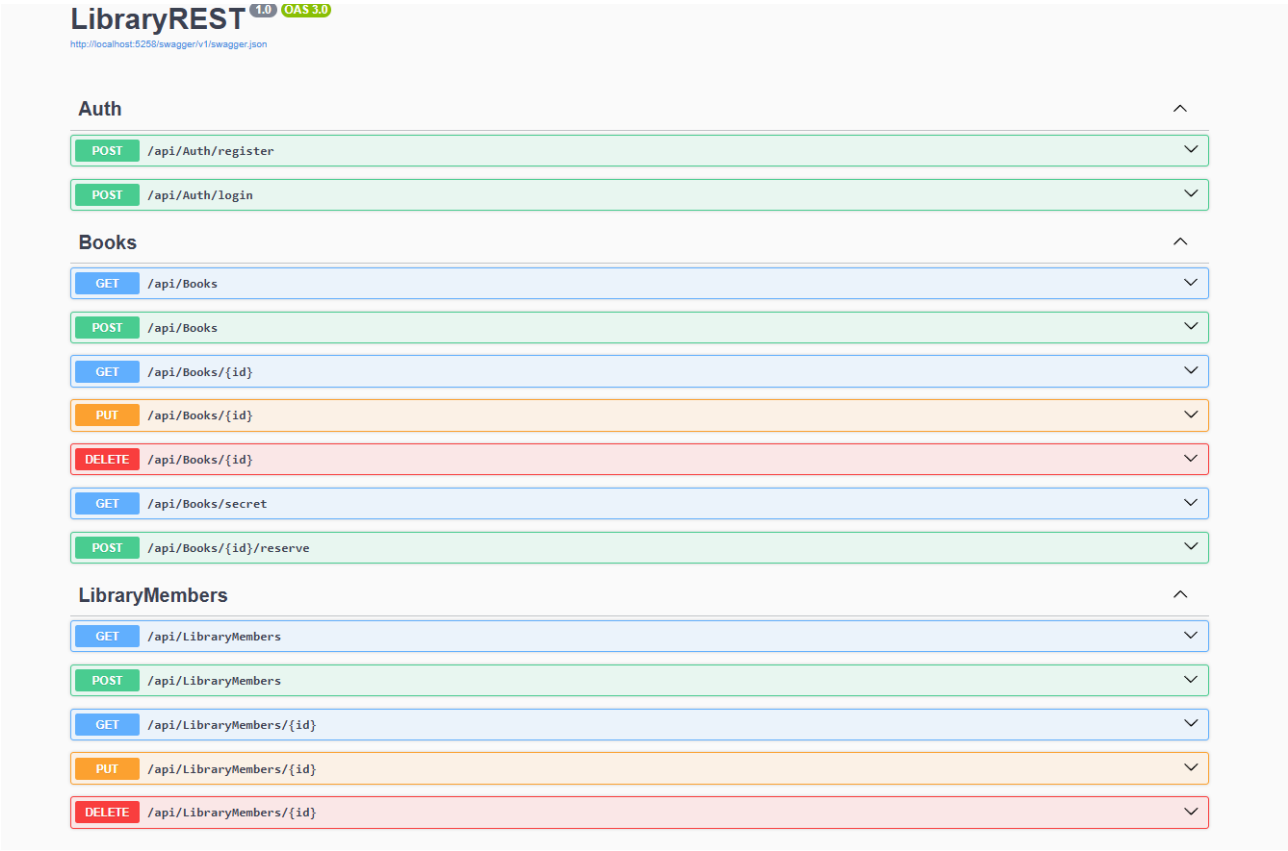




4. Модифікуйте створений у четвертій лабораторній REST API, щоб кінцеві точки використовували авторизацію, та надавали доступ відповідно до токену із заголовку HTTP-запиту Authorization. Зверніть увагу, щоб авторизація використовувалася лише в необхідних кінцевих точках, наприклад у REST API бібліотеки, перегляд книг не потребує авторизації, але бронювання книги потребує.

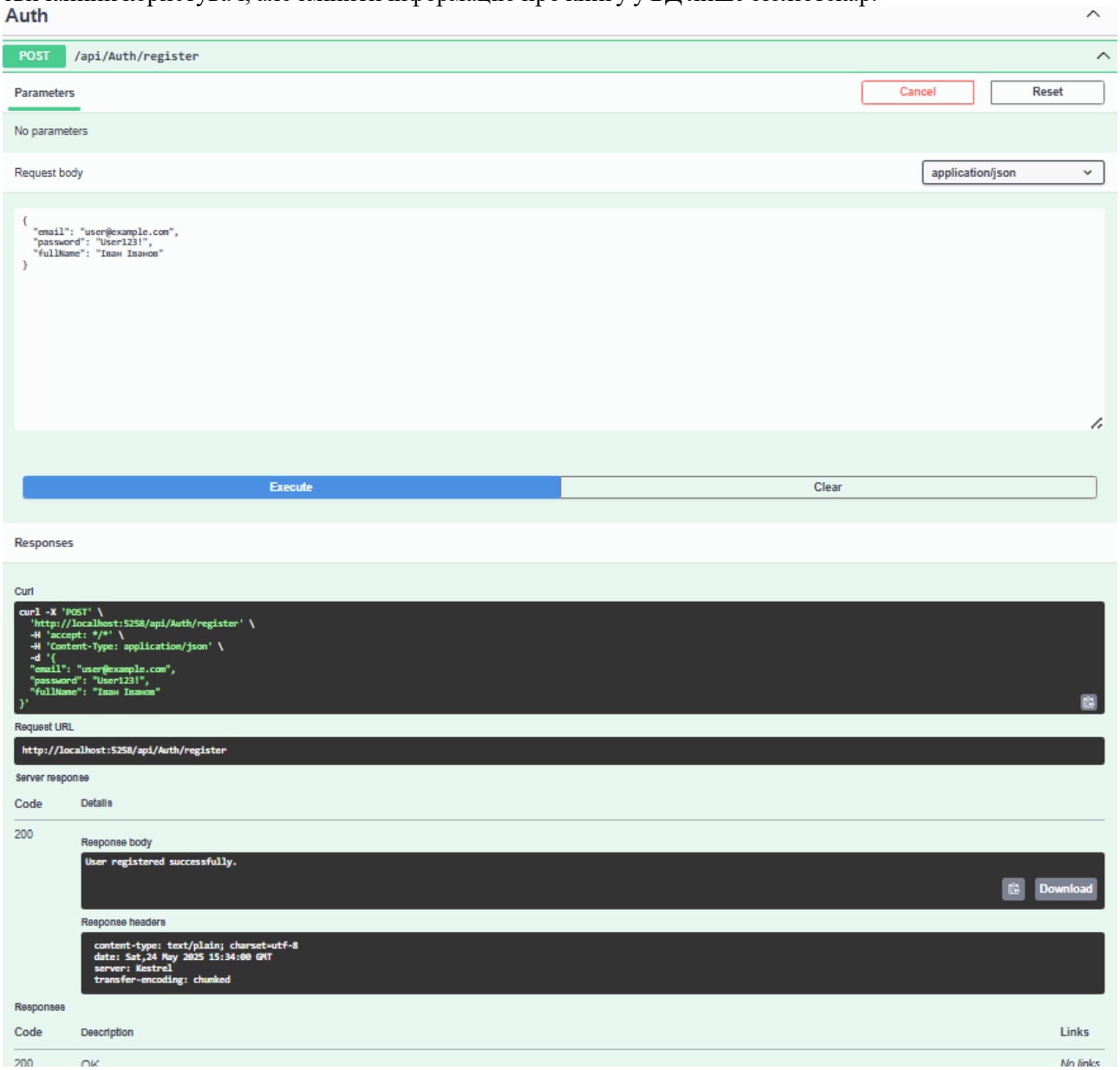
```
namespace LibraryREST.Controllers
{
    [ApiController]
    [Route("api/[controller]")]

    public class BooksController : ControllerBase
    {
        private readonly ICrudServiceAsync<BookModel> _bookService;


        public BooksController(ICrudServiceAsync<BookModel> bookService)
        {
            _bookService = bookService;
        }

        // 🔓 Публічний доступ
        [HttpGet]
        [AllowAnonymous]

        public async Task<ActionResult<IEnumerable<BookApiModel>>> GetAll()
        {
            var books = await _bookService.ReadAllAsync();
            var result = books.Select(book => new BookApiModel
            {
                Id = book.Id,
                Title = book.Title,
                Author = book.Author
            });

            return Ok(result);
        }

        // 🔓 Публічний доступ
        [HttpGet("{id:int}")]
        [AllowAnonymous]

        public async Task<ActionResult<BookApiModel>> Get(int id)
        {
            var book = await _bookService.ReadAsync(id);
            if (book == null)
                return NotFound();

            var dto = new BookApiModel
            {
                Id = book.Id,
                Title = book.Title,
                Author = book.Author
            };

            return Ok(dto);
        }

        // 🔒 Створення книги — авторизація обов'язкова
```

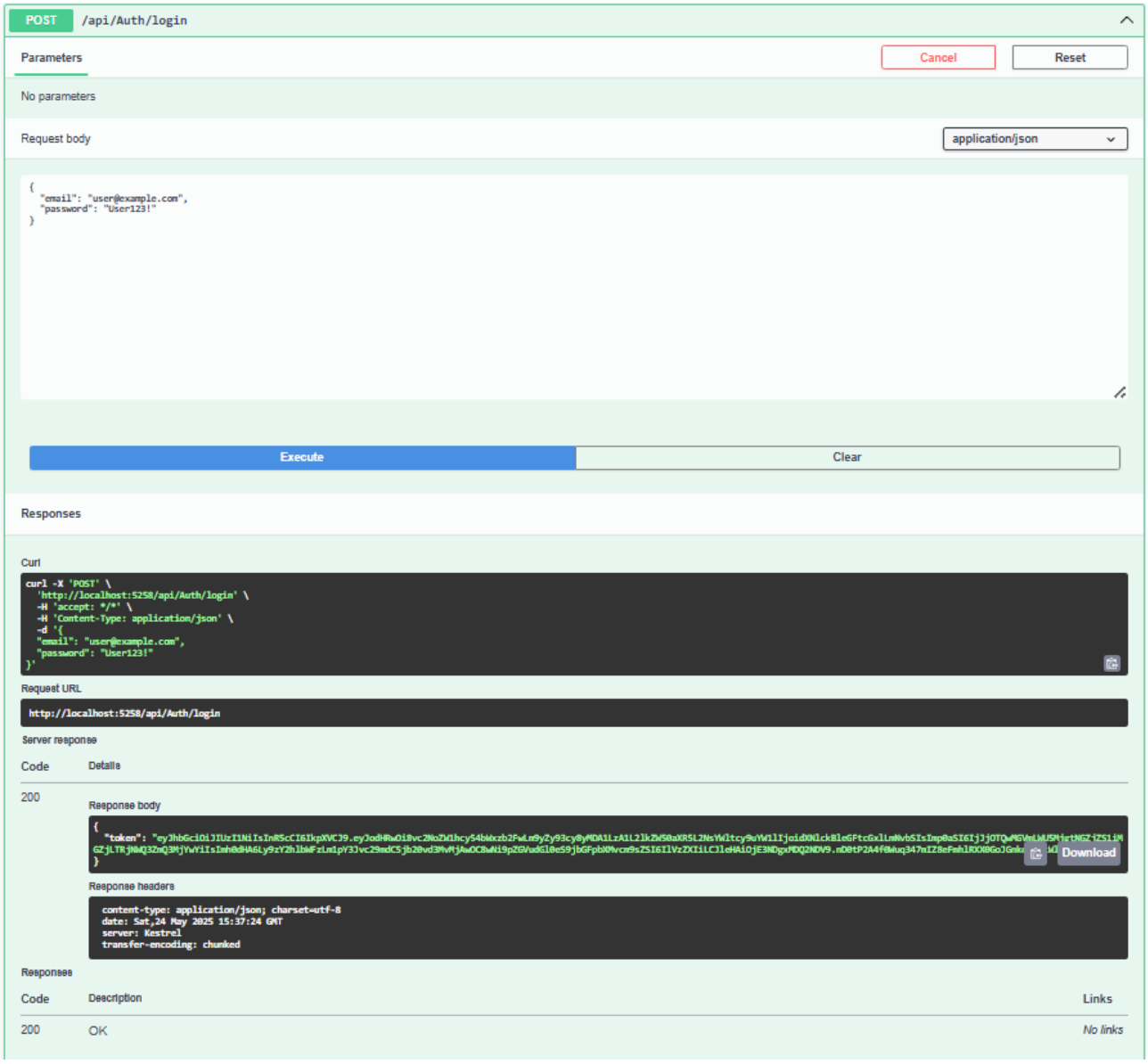
```
        [Authorize]
        Ссылок: 0
        public async Task<IActionResult> Create([FromBody] BookCreateModel mod
        {
            var newBook = new BookModel
            {
                Title = model.Title,
                Author = model.Author,
                LibraryMemberId = model.LibraryMemberId,
                BookTags = new List<BookTagModel>()
            };

            var created = await _bookService.CreateAsync(newBook);
            if (!created) return BadRequest("Could not create book.");

            await _bookService.SaveAsync();

            return CreatedAtAction(nameof(Get), new { id = newBook.Id }, newBo
        }

        // 🔒 Оновлення книги — авторизація обов'язкова
        [HttpPut("{id:int}")]
        [Authorize]
        Ссылок: 0
        public async Task<IActionResult> Update(int id, [FromBody] BookUpdateM
        {
            var book = await _bookService.ReadAsync(id);
            if (book == null)
                return NotFound();

            book.Title = model.Title;
            book.Author = model.Author;
            book.LibraryMemberId = model.LibraryMemberId;

            var updated = await _bookService.UpdateAsync(book);
            if (!updated) return BadRequest("Update failed.");

            await _bookService.SaveAsync();

            return NoContent();
        }

        // 🔒 Видалення книги — авторизація обов'язкова
        [HttpDelete("{id:int}")]
        [Authorize]
        Ссылок: 0
        public async Task<IActionResult> Delete(int id)
        {
            var book = await _bookService.ReadAsync(id);
            if (book == null)
                return NotFound();

            var deleted = await _bookService.RemoveAsync(book);
            if (!deleted) return BadRequest("Delete failed.");

            await _bookService.SaveAsync();

            return NoContent();
        }

        // 🔒 Тестовий/секретний ендпоінт
        [Authorize]
        [HttpGet("secret")]
        Ссылок: 0
        public IActionResult GetSecret()
        {
            return Ok("Тільки для авторизованих користувачів");
        }

        // 🔒 Умовний ендпоінт для бронювання книги
        [Authorize]
        [HttpPost("{id:int}/reserve")]
        Ссылок: 0
        public async Task<IActionResult> ReserveBook(int id)
        {
            var book = await _bookService.ReadAsync(id);
            if (book == null) return NotFound();

            // Тут можна реалізувати логіку "бронювання"
            return Ok($"Книга '{book.Title}' заброньована.");
        }
    }
}
```



5. Додайте для створеного вебдодатку ролі, усього ролей повино бути не менше трьох. Оновіть кінцеві точки REST API, що використовують авторизацію, щоб вони виконувалися залежно від ролі клієнта, наприклад у REST API бібліотеки, забронювати книгу може

звичайний користувач, але змінити інформацію про книгу у БД лише бібліотекар.



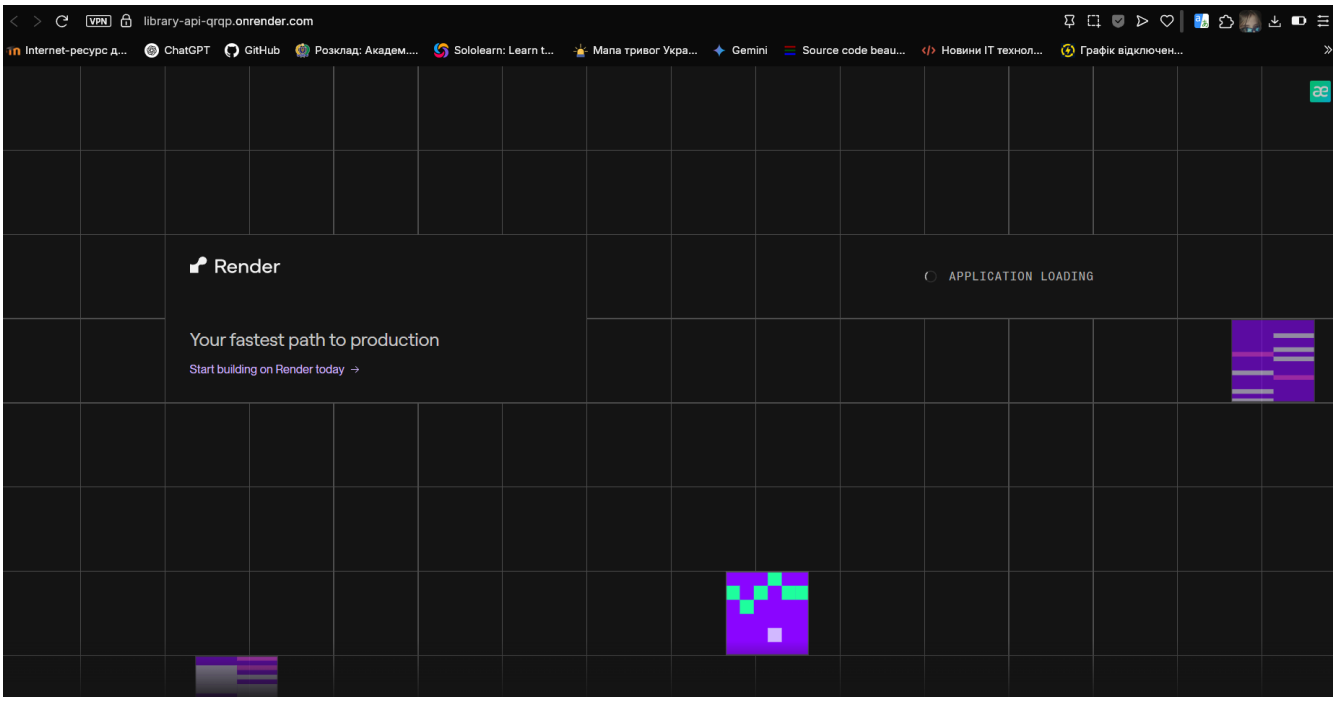


До готової лабораторної роботи додайте PDF файл у якому будуть результати виконання запитів, використовуючи модифікований REST API, який буде повертати результати реєстрації та аутентифікації, виконання запитів для авторизованих, неавторизованих та клієнтів, із недостатнім рівнем доступа. Можете скористатися сторінкою Swagger UI або застосунком Postman.

Додаткове завдання

Розгорніть ваш застосунок на одній із хостингових або хмарних платформ, як Azure, Heroku, AWS, GCP і т.п. У звіті приклади запитів повинні використовувати розгорнутий застосунок. До звіту також додайте скріншоти із розгорнутою інфраструктурою проєкта та посилання на застосунок.

Пробувала через Render.com – Відкрилась сторінка і хвилин 20 вона просто грузилась.


library-api-qrqp.onrender.com
Render
APPLICATION LOADING
Your fastest path to production
Start building on Render today →