МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Основы разработки корпоративных систем на платформе .NET»

Тема: «Разработка программного интерфейса для взаимодействия с приложением»

Студент гр. 6305 _____ Стрельников В.Е.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург

2020

Содержание

Цель работы

Разработать программный интерфейс веб-API приложения с помощью ASP.NET Core 3.1 в среде JetBrains Rider.

Основные положения

В процессе создания веб-API приложения «Библиотека книг» с помощью ASP.NET Core будет сделано следующее:

1. Создание проекта, состоящего из слоёв (Core layer, Business logic layer, Data Access layer), что позволит сделать их независимыми друг от друга и легко дополняемыми\изменяемыми;
2. Реализация паттернов Repository и Unit of Work;
3. Реализация использования Entity Framework для работы с базой данных SQL Server Express;
4. Реализация использования Automapper для маппаинга моделей в ресурсы веб-API;
5. Реализация использования Swagger для формирования удобного веб-интерфейса тестирования приложения с помощью запросов;
6. Будут реализованы Unit тесты слоя бизнес логики приложения.

В данной лабораторной работе рассматривается разработка программного интерфейса для взаимодействия с приложением.

Выполнение

На уровне представления (веб-API слой) мы будем обрабатывать вызовы API Restful, сопоставления ресурсов и проверки ресурсов. Добавим Swagger в проект, чтобы визуализировать результат. Для конфигурации пакета добавим в Startup следующие строки:

```
services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Book Library",
        Description = "This is the book library",
```

```
        Version = "v1"
    });
});
```

А в методе Configure:

```
app.UseSwagger();

app.UseSwaggerUI(options =>
{
    options.RoutePrefix = "";
    options.SwaggerEndpoint("/swagger/v1/swagger.json", "Book Library V1");
});
```

Здесь мы указываем название для представления и создаём папку с версией №1.

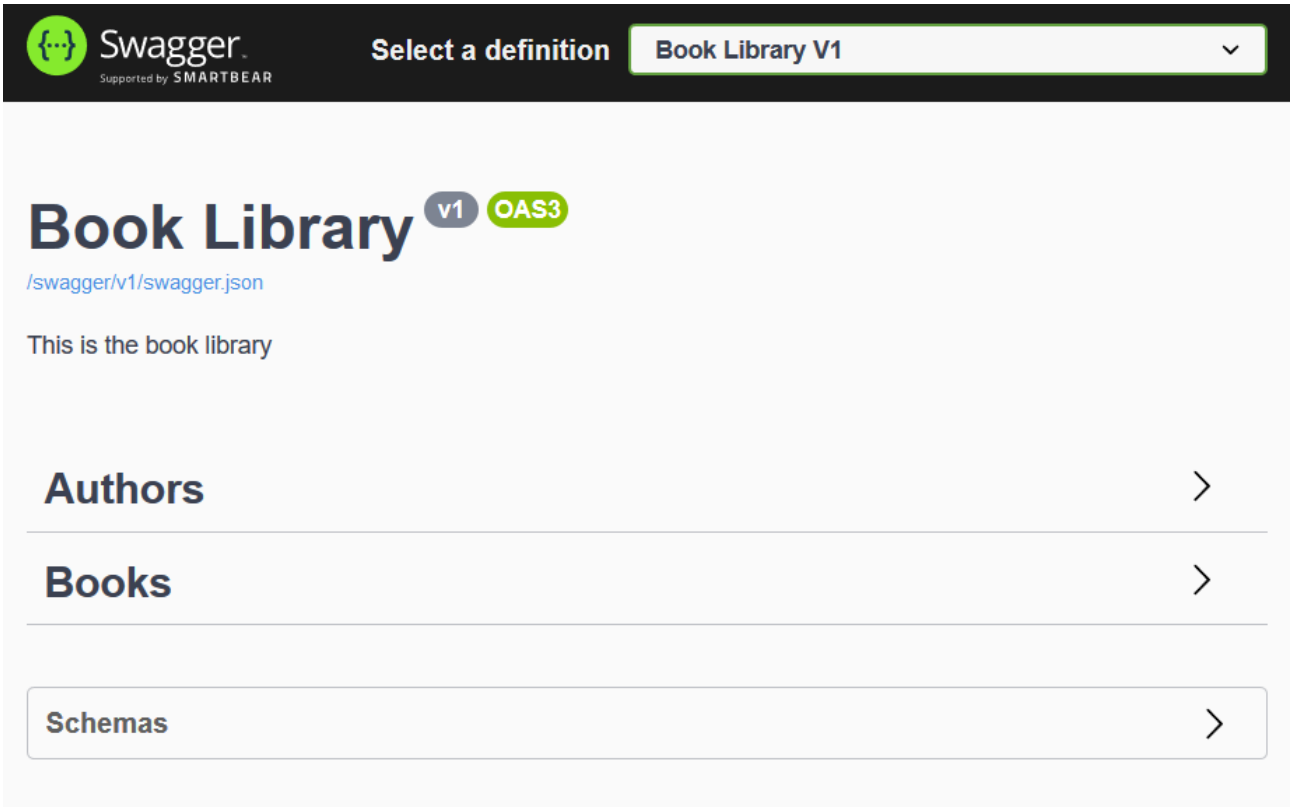

Рисунок 1. Демонстрация запуска приложения с активным Swagger.

Теперь необходимо перейти непосредственно к созданию контроллеров.

Описывать будем следующий API-интерфейс.

API	Описание	Текст запроса	Текст ответа
GET	Получение всех элементов	Отсутствует	Массив элементов

GET /{id}	Получение объекта по идентификатору	Отсутствует	Элемент
POST	Добавление нового элемента	Элемент	Элемент
PUT /{id}	Обновление существующего элемента	Элемент	Отсутствует
DELETE	Удаление элемента	Отсутствует	Отсутствует

Для маппинга будем использовать AutoMapper. Принцип маппинга укажем в файле конфигурации, его вид будет следующий:

```
namespace Book.API.Mapping
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            // Domain to Resource
            CreateMap<BookModel, BookResource>();
            CreateMap<AuthorModel, AuthorResource>();
            CreateMap<BookModel, SaveBookResource>();
            CreateMap<AuthorModel, SaveAuthorResource>();

            // Resource to Domain
            CreateMap<BookResource, BookModel>();
            CreateMap<AuthorResource, AuthorModel>();
            CreateMap<SaveBookResource, BookModel>();
            CreateMap<SaveAuthorResource, AuthorModel>();

        }
    }
}
```

Контроллеры книг и авторов с реализованными запросами будут выглядеть следующим образом:

```
namespace Book.API.Controllers
{
    [Route("api/[controller]")]
```

```
    [ApiController]
    public class BooksController : ControllerBase
    {
        private readonly IBookService _bookService;
        private readonly IMapper _mapper;

        public BooksController(IBookService bookService, IMapper mapper)
        {
            this._mapper = mapper;
            this._bookService = bookService;
        }

        [HttpGet("")]
        public async Task<OkObjectResult> GetAllBooks()
        {
            var books = await _bookService.GetAllWithAuthor();
            var bookResources =
                _mapper.Map<IEnumerable<BookModel>, IEnumerable<BookResource>>(books);

            return Ok(bookResources);
        }

        [HttpGet("{id}")]
        public async Task<OkObjectResult> GetBookById(int id)
        {
            var book = await _bookService.GetBookById(id);
            var bookResource = _mapper.Map<BookModel, BookResource>(book);

            return Ok(bookResource);
        }

        [HttpPost("")]
        public async Task<ActionResult<BookResource>> CreateBook([FromBody]
SaveBookResource saveBookResource)
        {
            var validator = new SaveBookResourceValidator();
            var validationResult = await validator.ValidateAsync(saveBookResource);

            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }

            var bookToCreate = _mapper.Map<SaveBookResource,
BookModel>(saveBookResource);

            var newBook = await _bookService.CreateBook(bookToCreate);

            var book = await _bookService.GetBookById(newBook.Id);

            var bookResource = _mapper.Map<BookModel, BookResource>(book);

            return Ok(bookResource);
        }

        [HttpPut("{id}")]
        public async Task<ActionResult<BookResource>> UpdateBook(int id, [FromBody]
SaveBookResource saveBookResource)
        {
            var validator = new SaveBookResourceValidator();
```

```
            var validationResult = await validator.ValidateAsync(saveBookResource);

            var requestIsInvalid = id == 0 || !validationResult.IsValid;

            if (requestIsInvalid)
            {
                return BadRequest(validationResult.Errors);
            }

            var bookToBeUpdate = await _bookService.GetBookById(id);

            if (bookToBeUpdate == null)
            {
                return NotFound();
            }

            var book = _mapper.Map<SaveBookResource, BookModel>(saveBookResource);

            await _bookService.UpdateBook(bookToBeUpdate, book);

            var updatedBook = await _bookService.GetBookById(id);
            var updatedBookResource = _mapper.Map<BookModel,
BookResource>(updatedBook);

            return Ok(updatedBookResource);
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteBook(int id)
        {
            if (id == 0)
            {
                return BadRequest();
            }

            var book = await _bookService.GetBookById(id);

            if (book == null)
            {
                return NotFound();
            }

            await _bookService.DeleteBook(book);

            return NoContent();
        }
    }
}


namespace Book.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthorsController : ControllerBase
    {
        private readonly IAuthorService _authorService;
        private readonly IMapper _mapper;
```

```
        public AuthorsController(IAuthorService authorService, IMapper mapper)
        {
            this._mapper = mapper;
            this._authorService = authorService;
        }

        [HttpGet("")]
        public async Task<OkObjectResult> GetAllAuthors()
        {
            var authors = await _authorService.GetAllAuthors();
            var authorResources = _mapper.Map<IEnumerable<AuthorModel>,
IEnumerable<AuthorResource>>(authors);

            return Ok(authorResources);
        }

        [HttpGet("{id}")]
        public async Task<OkObjectResult> GetAuthorById(int id)
        {
            var author = await _authorService.GetAuthorById(id);
            var authorResource = _mapper.Map<AuthorModel, AuthorResource>(author);

            return Ok(authorResource);
        }

        [HttpPost("")]
        public async Task<ActionResult<AuthorResource>> CreateAuthor([FromBody]
SaveAuthorResource saveAuthorResource)
        {
            var validator = new SaveAuthorResourceValidator();
            var validationResult = await validator.ValidateAsync(saveAuthorResource);

            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }

            var authorToCreate = _mapper.Map<SaveAuthorResource,
AuthorModel>(saveAuthorResource);

            var newAuthor = await _authorService.CreateAuthor(authorToCreate);

            var author = await _authorService.GetAuthorById(newAuthor.Id);

            var authorResource = _mapper.Map<AuthorModel, AuthorResource>(author);

            return Ok(authorResource);
        }

        [HttpPut("{id}")]
        public async Task<ActionResult<AuthorResource>> UpdateAuthor(int id,
            [FromBody] SaveAuthorResource saveAuthorResource)
        {
            var validator = new SaveAuthorResourceValidator();
            var validationResult = await validator.ValidateAsync(saveAuthorResource);

            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
```

```
            var authorToBeUpdated = await _authorService.GetAuthorById(id);

            if (authorToBeUpdated == null)
            {
                return NotFound();
            }

            var author = _mapper.Map<SaveAuthorResource,
AuthorModel>(saveAuthorResource);

            await _authorService.UpdateAuthor(authorToBeUpdated, author);

            var updatedAuthor = await _authorService.GetAuthorById(id);

            var updatedAuthorResource = _mapper.Map<AuthorModel,
AuthorResource>(updatedAuthor);

            return Ok(updatedAuthorResource);
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteAuthor(int id)
        {
            var author = await _authorService.GetAuthorById(id);

            await _authorService.DeleteAuthor(author);

            return NoContent();
        }
    }
}
```

Используя Swagger, проверим работоспособность запросов.

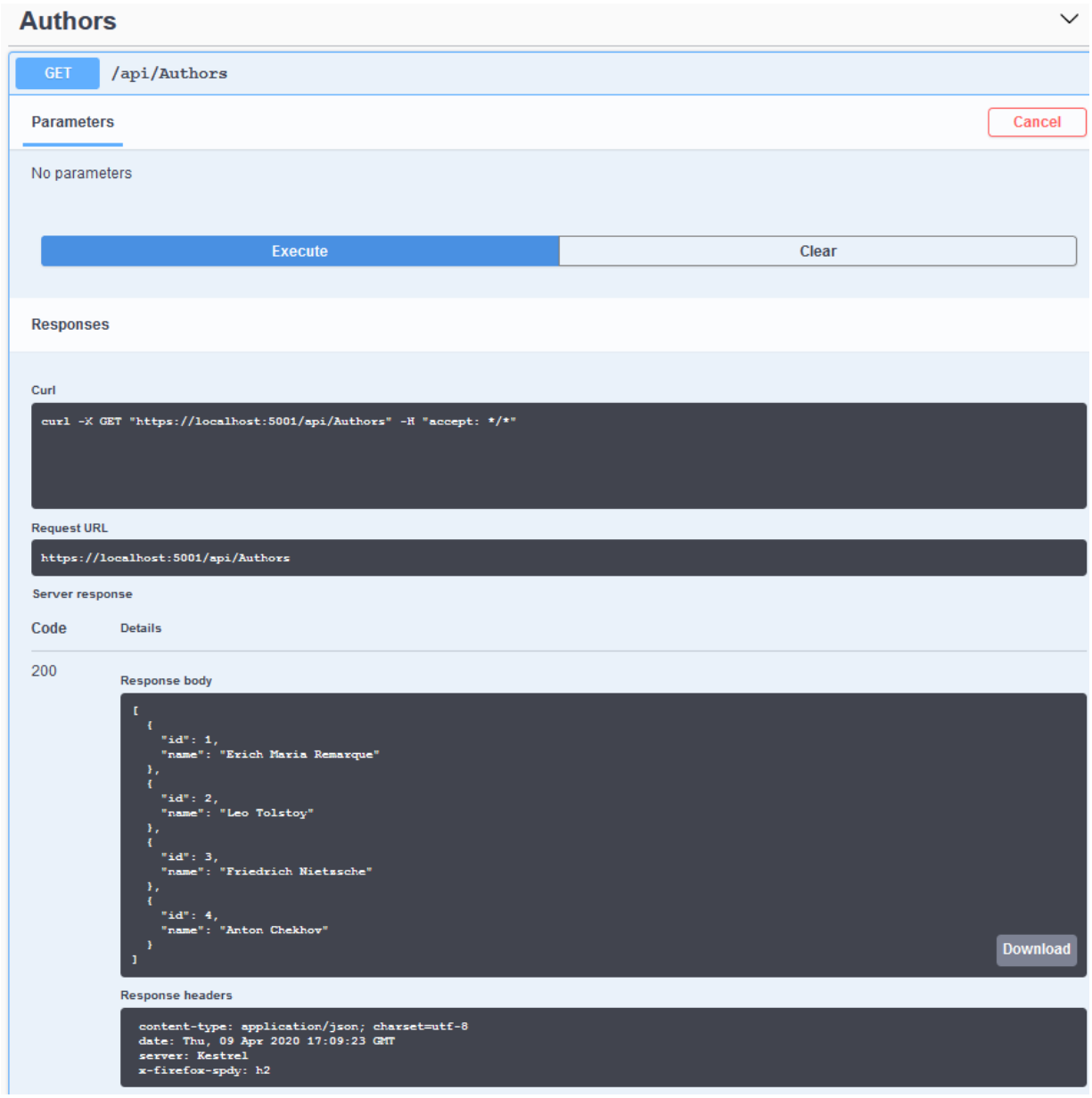

Рисунок 1. GET запрос для авторов. В результате получаем список авторов.

Аналогичный запрос можно выполнить и для книг.

Проверим DELETE запрос и удалим из базы следующую книгу:

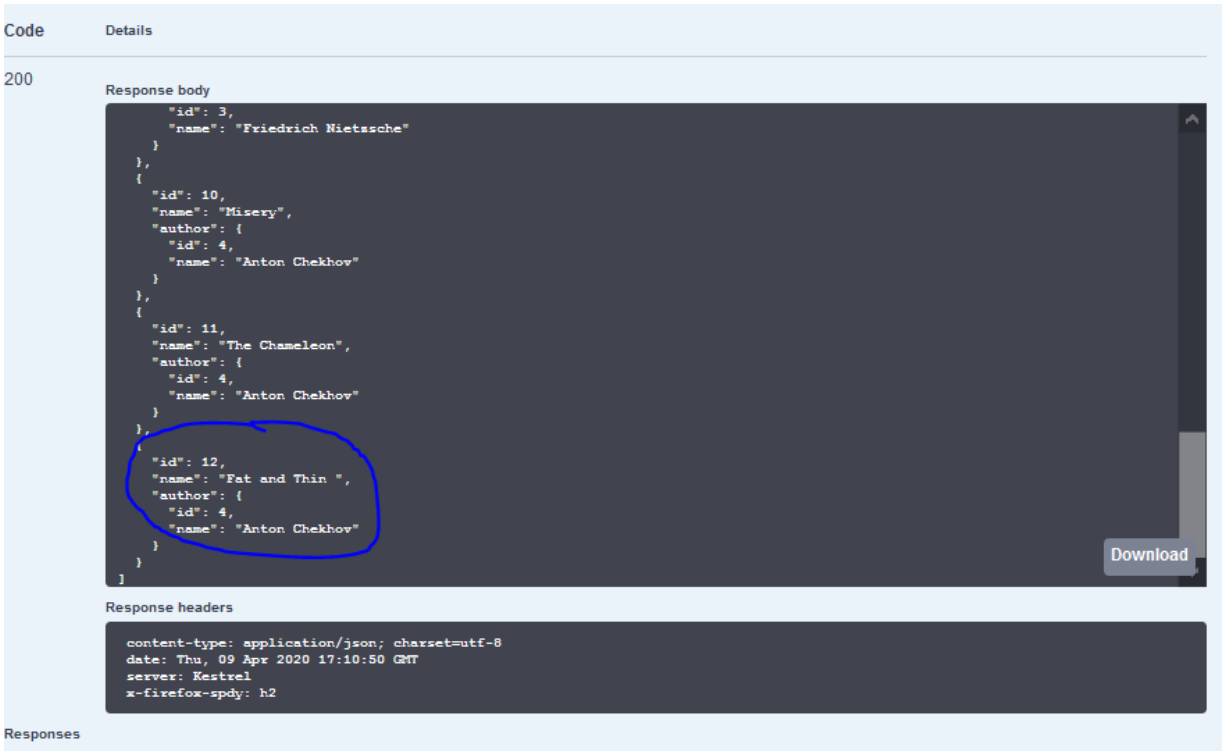

Рисунок 2. Список книг до удаления выбранной книги.

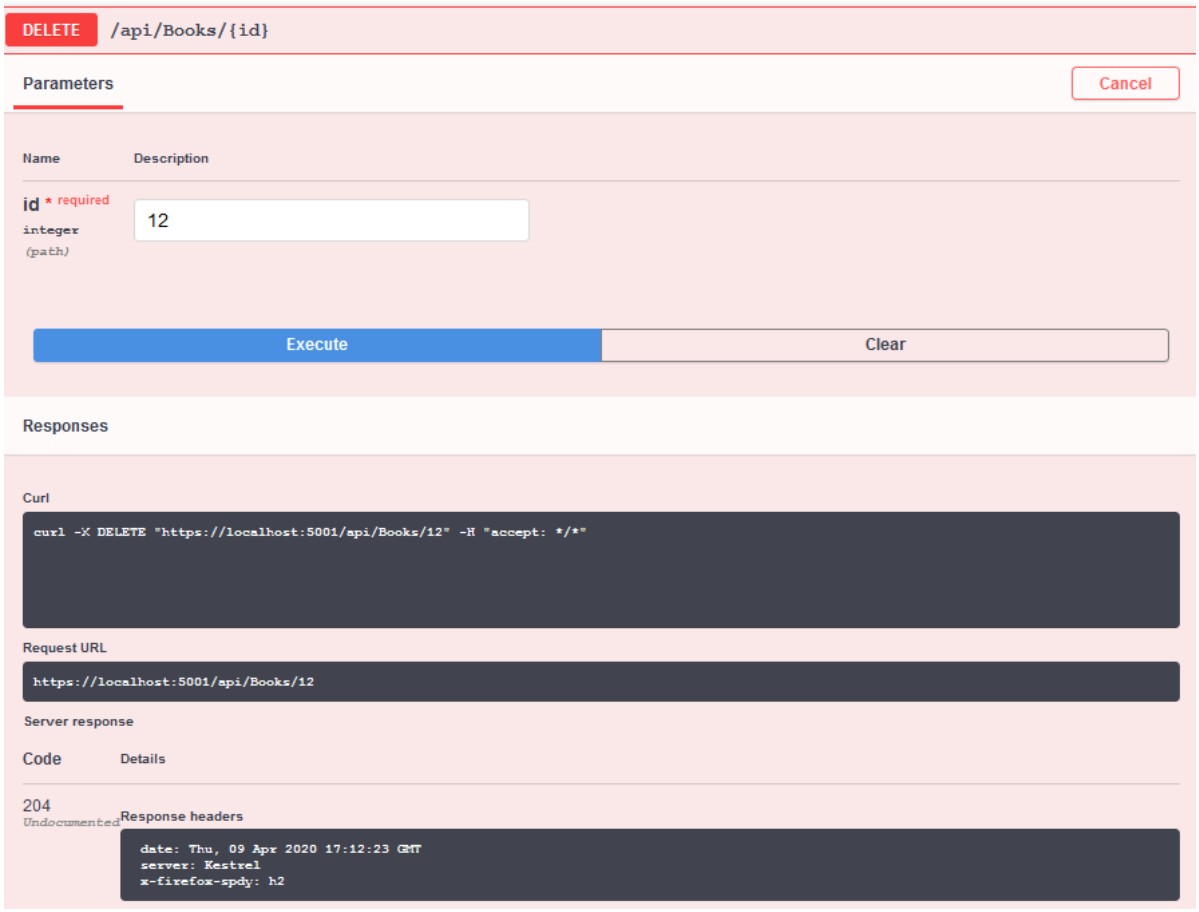

Рисунок 3. Вызов запроса DELETE с передачей id удаляемой книги.

Убеждаемся в том, что при вызове GET запроса выбранная нами книга больше не будет существовать.

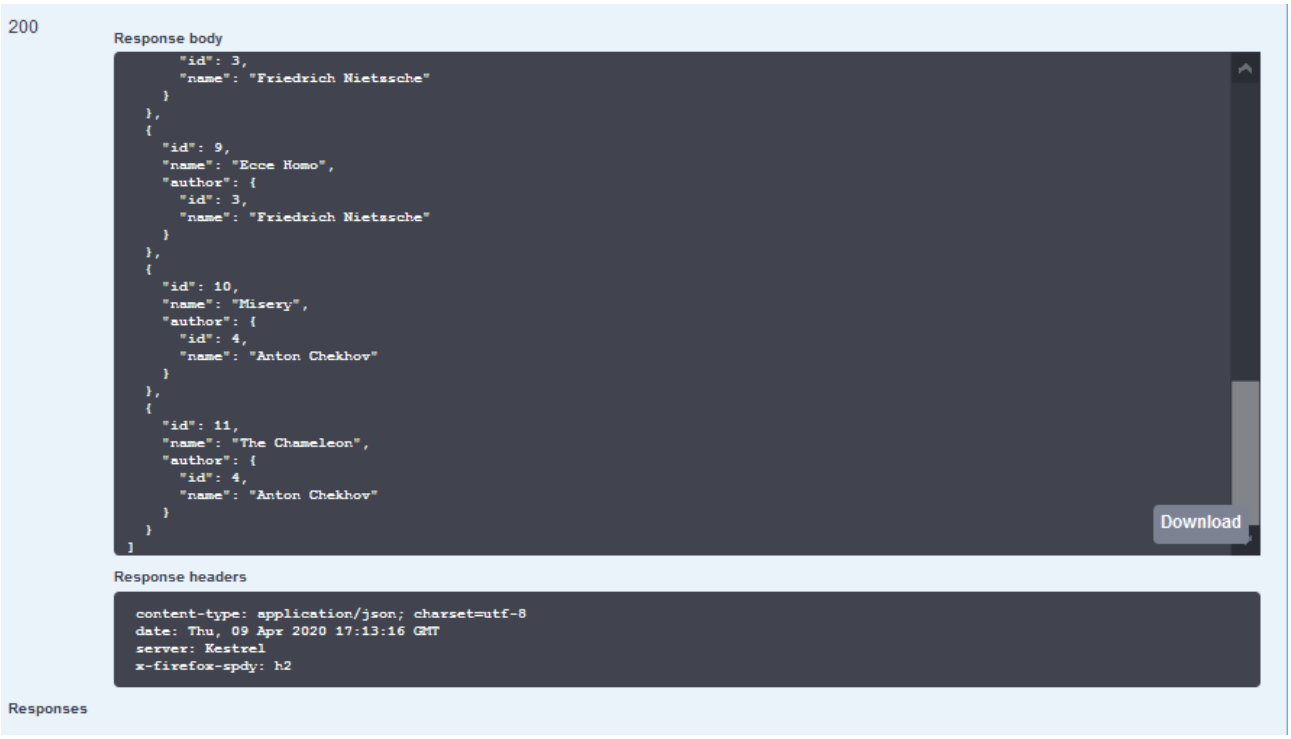

Рисунок 4. Вывод списка книг после удаления.

Аналогичным образом работают и другие запросы обоих контроллеров.

Таким образом уровень API будет отвечать за представление и получение запросов для приложения. Это будет первая проверка ресурсов, которые не позволят приложению получать нежелательные данные. Это уровень, который сопоставляет ресурсы с моделями.

Вывод

В процессе выполнения данной лабораторной работы был разработан программный интерфейс веб-API приложения «Библиотека книг» с помощью ASP.NET Core 3.1 в среде JetBrains Rider.