**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА №** 2

**Дисциплина: «Backend»**

**Тема:** *Создание приложения на основе класса WebApplication на основе ASP.NET Core 2*

**Выполнил: студент группы: 231-339**

Карапетян Нвер Каренович\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О.)



**Дата, подпись:** 09.03.25 \_\_\_\_\_\_\_\_\_\_\_\_

(Дата)(Подпись)

**Проверил:** \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_*

(Фамилия И.О., степень, звание) (Оценка)

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_*

(Дата)(Подпись)

**Москва**

**2025**

Цель:

Ознакомиться с базовыми шагами создания веб-приложения на основе класса WebApplication в ASP.NET Core.

Инструменты и технологии:

* **ASP.NET Core** – фреймворк для разработки веб-приложений.
* **Entity Framework Core** – ORM для работы с базой данных.
* **SSMS** – инструмент для управления базами данных (MS SQL или MySQL).
* **Scalar** – дополнительный инструмент для тестирования API.

Ход работы

Целью данной лабораторной работы было создание backend-приложения финансового проекта, в котором можно получать информацию об акциях популярных компаний (цена, общая капитализация и др.) и добавлять акции в свое портфолио. Проект реализован с использованием ASP.NET Core и Entity Framework. В процессе разработки были рассмотрены ключевые аспекты работы с базой данных: создание моделей, миграции, связи между таблицами, а также создание репозиториев и реализация взаимодействия с внешним API.

Шаг 1: Создание моделей на C#

На первом этапе разработки были созданы модели данных на языке C#. Модели отражают структуру таблиц в базе данных и включают необходимые свойства и типы данных. Каждая модель была спроектирована с учетом логики предметной области.

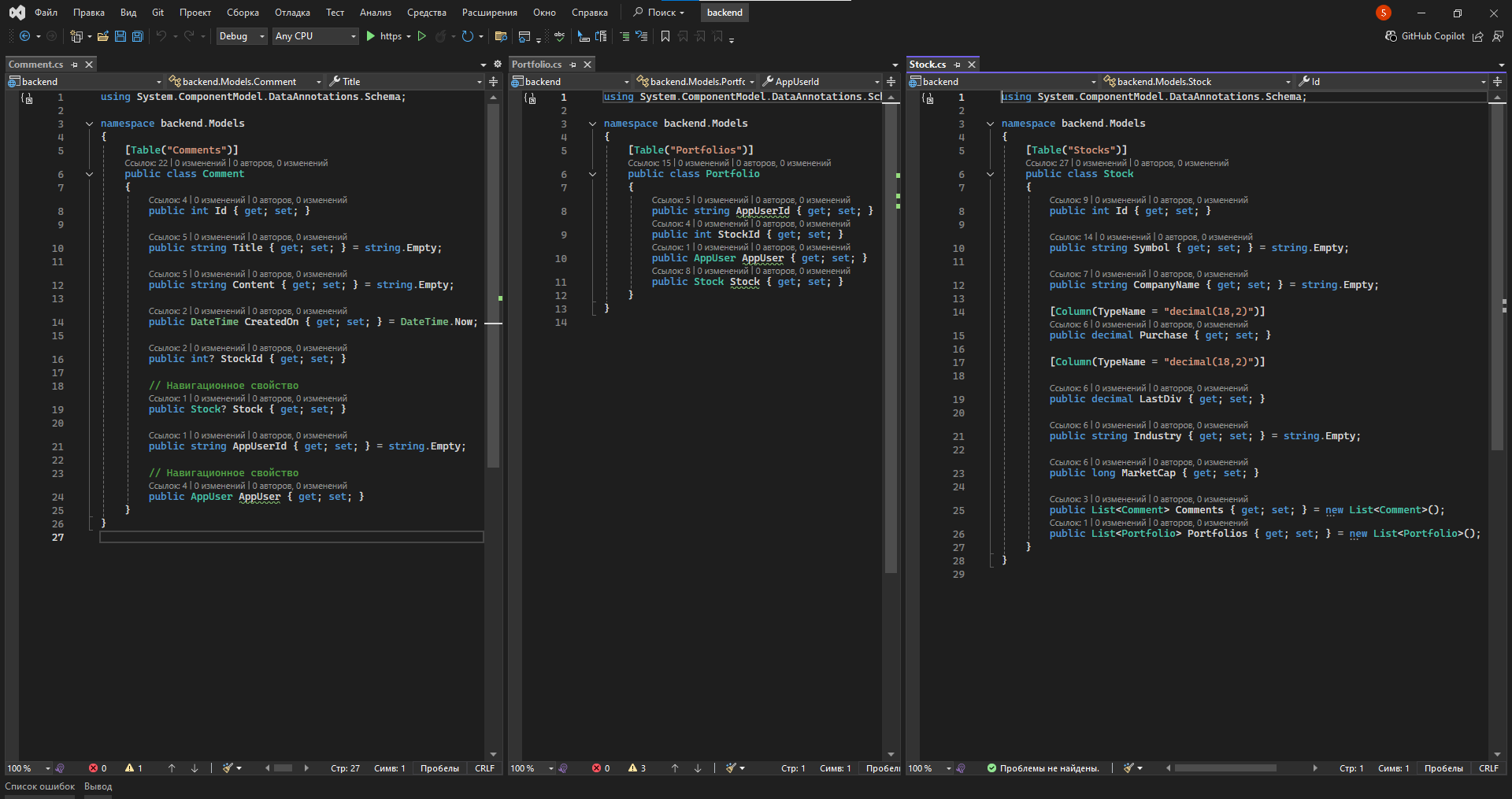


Рисунок 1. Модели таблиц комментариев, портфолио и самих акций соответственно.

На этом же этапе важно выстроить верно типы связей между таблицами посредством внедрения внешних ключей в сами модели. Там, где нам необходима единичная связь, были добавлены свойства, которые хранили в себе уникальный идентификатор экземпляра того типа, к которому ссылается данный ключ (пример: свойство с уникальным идентификатором в таблице Comments «public int? StockId { get; set; }» ссылается на акцию). Там, где нам необходима множественная связь, добавляем свойство, представляющее из себя список (пример: свойство «public List<Comment> Comments { get; set; } = new List<Comment>();» у модели Stock).

Также были добавлены навигационные свойства для удобства работы с моделями данных.

Шаг 2: Миграции и обновление базы данных

После создания моделей был использован подход Code-First для генерации базы данных. С помощью команды Add-Migration в консоли диспетчера пакетов NuGet (предварительно нужно установить в проекте библиотеку EntityFramework.Tools) был сгенерирован файл миграции, который содержал инструкции по созданию таблиц и других объектов базы данных.

Команда для создания миграции: «Add-Migration Init», где «Init» — название миграции.

Затем, для применения миграции и синхронизации базы данных с моделями, была использована команда «Update-Database», после которой в базе данных создавались таблицы по образу и подобию созданных ранее моделей.

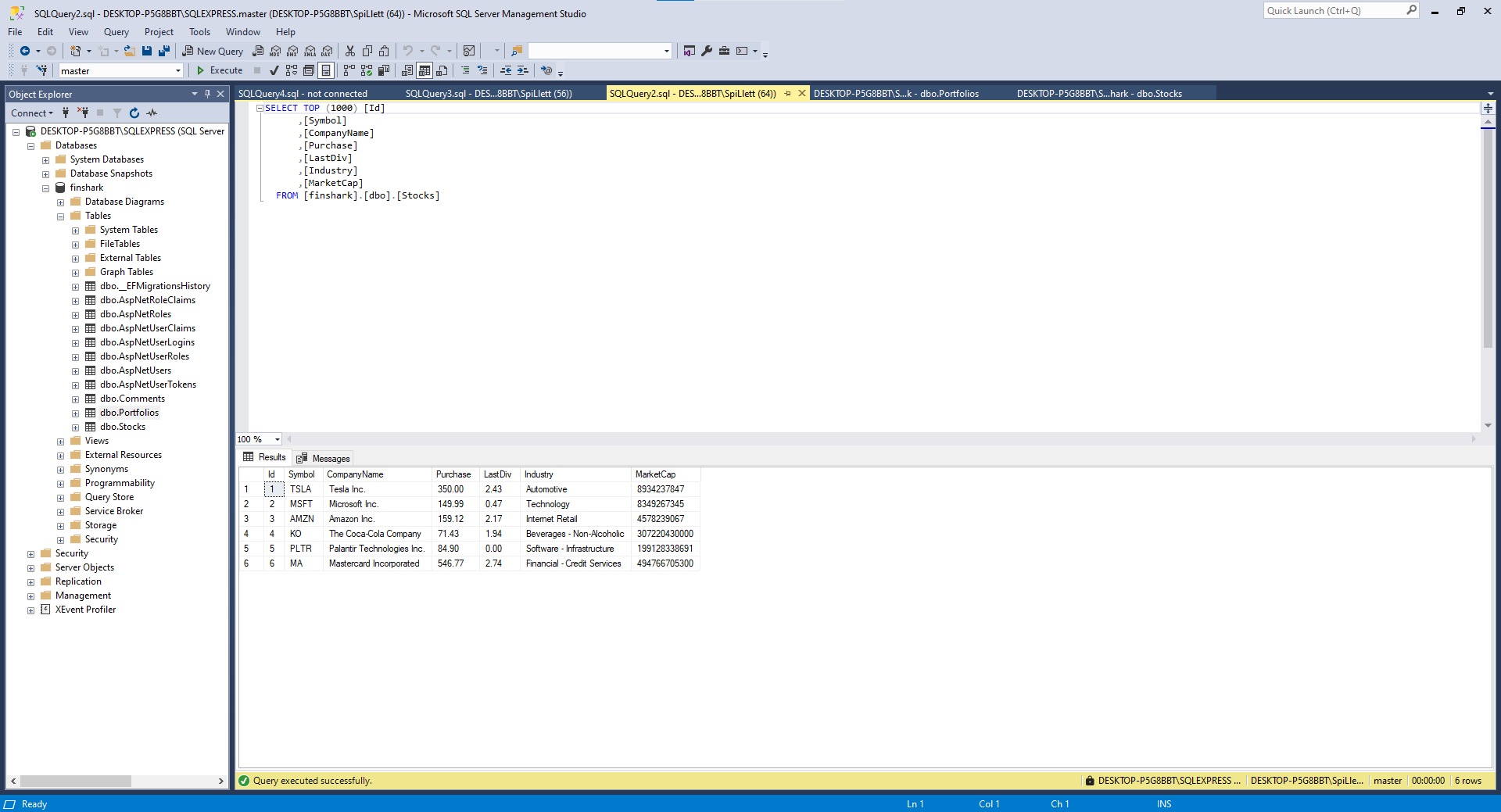


Рисунок 2. Таблицы в базе данных.

Шаг 4: Создание DTO

Для обмена данными между слоями приложения были созданы объекты передачи данных (DTO). DTO представляют собой упрощенные структуры, которые скрывают детали внутренней реализации и представляют только те данные, которые необходимы для выполнения операций.

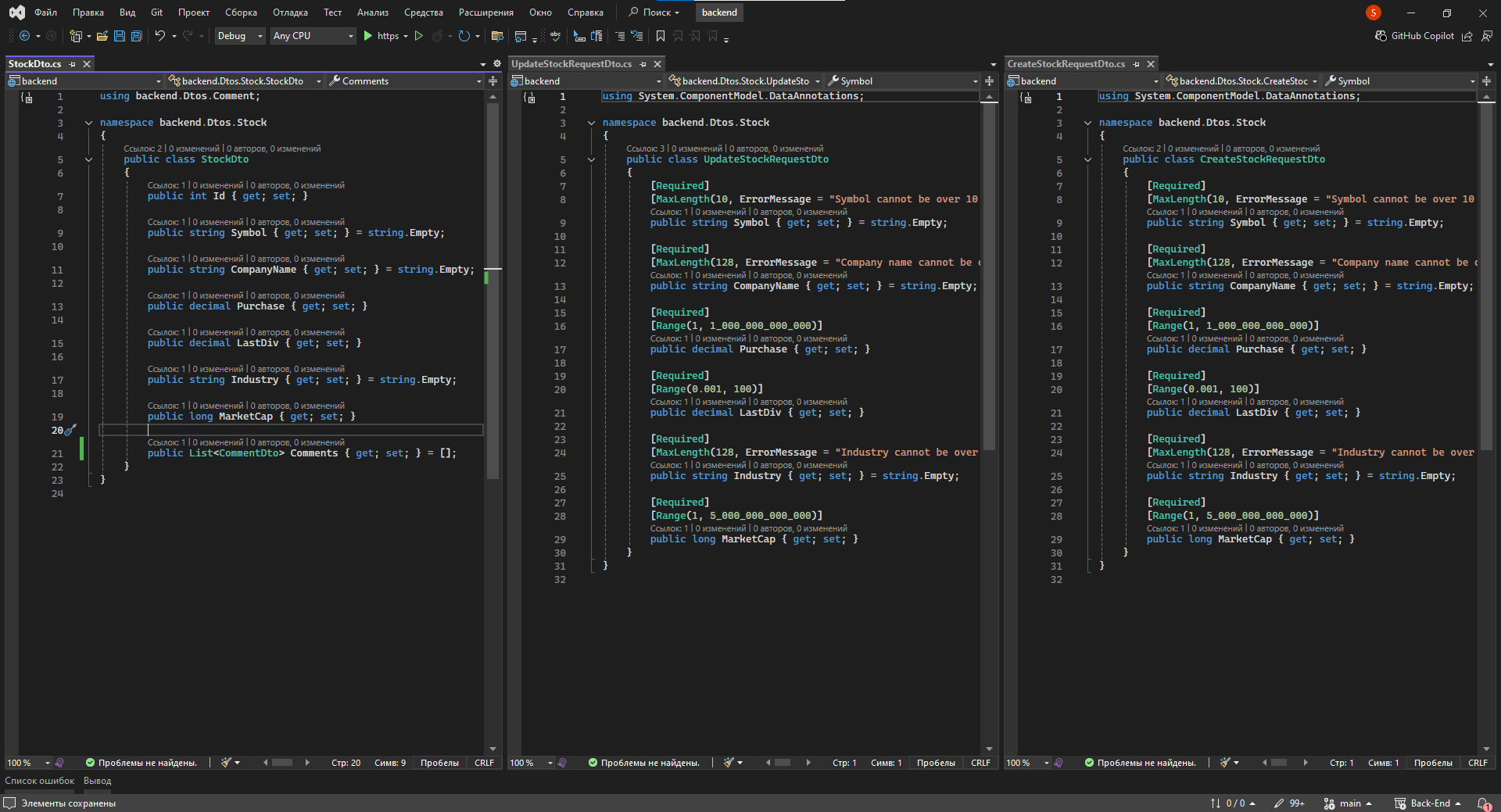


Рисунок 3. Примеры созданных DTO-классов для работы с моделью акций.

Таким образом были созданы DTO для GET, POST и PUT методов с валидацией данных с помощью атрибутов (Required, Range и др.).

Шаг 5: Интерфейсы репозиториев

Для работы с данными была реализована архитектура с использованием репозиториев. Созданы интерфейсы репозиториев для выполнения базовых операций с базой данных (CRUD). Каждый интерфейс содержит методы для получения, добавления, обновления и удаления данных.

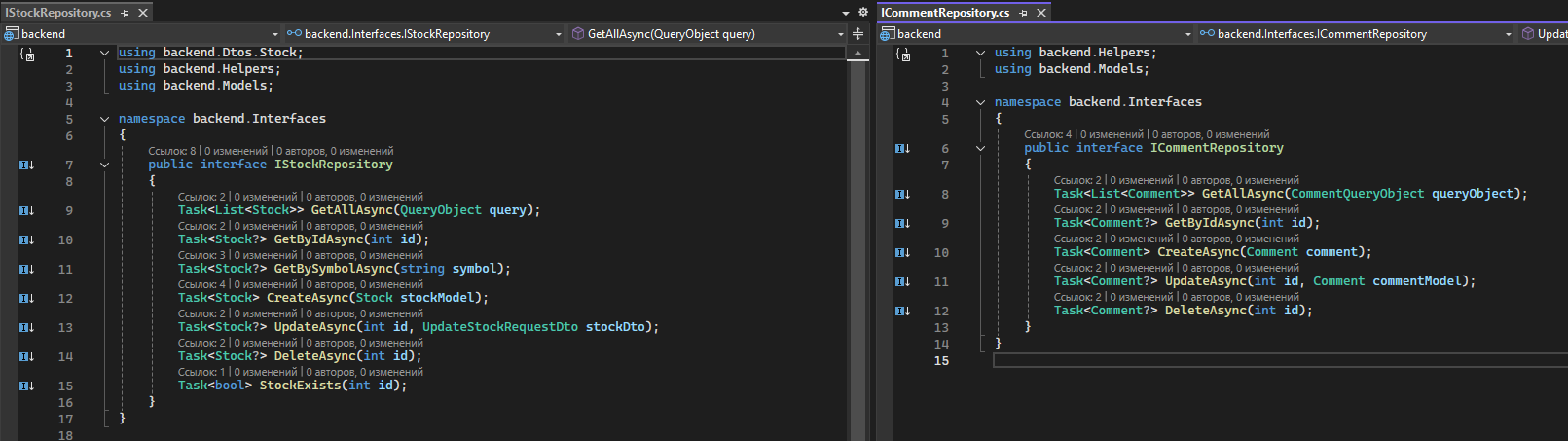


Рисунок 4. Пара примеров интерфейсов репозиториев.

Шаг 6: Реализация репозиториев

На основе интерфейсов были созданы реализации репозиториев, которые инкапсулируют логику взаимодействия с базой данных. Репозитории используют Entity Framework для выполнения операций с данными.

Пример реализации репозитория для работы с моделью Stock:

using backend.Data;

using backend.Dtos.Stock;

using backend.Helpers;

using backend.Interfaces;

using backend.Models;

using Microsoft.EntityFrameworkCore;

namespace backend.Repository

{

    public class StockRepository : IStockRepository

    {

        private readonly ApplicationDbContext \_context;

        public StockRepository(ApplicationDbContext context)

        {

            \_context = context;

        }

        public async Task<List<Stock>> GetAllAsync(QueryObject query)

        {

            var stocks = \_context.Stocks.Include(c => c.Comments).ThenInclude(a => a.AppUser).AsQueryable();

            if (!string.IsNullOrWhiteSpace(query.CompanyName))

                stocks = stocks.Where(s => s.CompanyName.Contains(query.CompanyName));

            if (!string.IsNullOrWhiteSpace(query.Symbol))

                stocks = stocks.Where(s => s.Symbol.Contains(query.Symbol));

            if (!string.IsNullOrWhiteSpace(query.SortBy))

            {

                if (query.SortBy.Equals("Symbol", StringComparison.OrdinalIgnoreCase))

                {

                    stocks = query.IsDescending

                        ? stocks.OrderByDescending(s => s.Symbol)

                        : stocks.OrderBy(s => s.Symbol);

                }

            }

            var skipNumber = (query.PageNumber - 1) \* query.PageSize;

            return await stocks

                .Skip(skipNumber)

                .Take(query.PageSize)

                .ToListAsync();

        }

        public async Task<Stock?> GetByIdAsync(int id)

        {

            return await \_context.Stocks

                .Include(s => s.Comments)

                .FirstOrDefaultAsync(s => s.Id == id);

        }

        public async Task<Stock> CreateAsync(Stock stockModel)

        {

            await \_context.Stocks.AddAsync(stockModel);

            await \_context.SaveChangesAsync();

            return stockModel;

        }

        public async Task<Stock?> DeleteAsync(int id)

        {

            var stockModel = await \_context.Stocks.FirstOrDefaultAsync(s => s.Id == id);

            if (stockModel is null)

                return null;

            \_context.Stocks.Remove(stockModel);

            await \_context.SaveChangesAsync();

            return stockModel;

        }

        public async Task<Stock?> UpdateAsync(int id, UpdateStockRequestDto stockDto)

        {

            var existingStock = await \_context.Stocks.FirstOrDefaultAsync(s => s.Id == id);

            if (existingStock is null)

                return null;

            existingStock.Symbol = stockDto.Symbol;

            existingStock.CompanyName = stockDto.CompanyName;

            existingStock.Purchase = stockDto.Purchase;

            existingStock.LastDiv = stockDto.LastDiv;

            existingStock.Industry = stockDto.Industry;

            existingStock.MarketCap = stockDto.MarketCap;

            await \_context.SaveChangesAsync();

            return existingStock;

        }

        public async Task<bool> StockExists(int id)

        {

            return await \_context.Stocks.AnyAsync(s => s.Id == id);

        }

        public async Task<Stock?> GetBySymbolAsync(string symbol)

        {

            return await \_context.Stocks.FirstOrDefaultAsync(s => s.Symbol == symbol);

        }

    }

}

Шаг 7: Реализация CRUD-запросов в контроллерах

Для взаимодействия с пользователями были реализованы контроллеры, которые выполняют операции CRUD, используя репозитории.

Пример контроллера StockController.cs:

using backend.Dtos.Stock;

using backend.Helpers;

using backend.Interfaces;

using backend.Mappers;

using Microsoft.AspNetCore.Mvc;

namespace backend.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class StockController : ControllerBase

    {

        private readonly IStockRepository \_stockRepo;

        public StockController(IStockRepository stockRepo)

        {

            \_stockRepo = stockRepo;

        }

        [HttpGet]

        public async Task<IActionResult> GetAll([FromQuery] QueryObject query)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

            var stocks = await \_stockRepo.GetAllAsync(query);

            var stocksDto = stocks.Select(s => s.ToStockDto()).ToList();

            return Ok(stocksDto);

        }

        [HttpGet("{id:int}")]

        public async Task<IActionResult> GetById([FromRoute] int id)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

            var stock = await \_stockRepo.GetByIdAsync(id);

            if (stock is null)

                return NotFound();

            return Ok(stock.ToStockDto());

        }

        [HttpPost]

        public async Task<IActionResult> Create([FromBody] CreateStockRequestDto stockDto)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

            var stockModel = stockDto.ToStockFromCreateDTO();

            await \_stockRepo.CreateAsync(stockModel);

            return CreatedAtAction(nameof(GetById), new { id = stockModel.Id }, stockModel.ToStockDto());

        }

        [HttpPut("{id:int}")]

        public async Task<IActionResult> Update([FromRoute] int id, [FromBody] UpdateStockRequestDto updateDto)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

            var stockModel = await \_stockRepo.UpdateAsync(id, updateDto);

            if (stockModel is null)

                return NotFound();

            return Ok(stockModel.ToStockDto());

        }

        [HttpDelete("{id:int}")]

        public async Task<IActionResult> Delete([FromRoute] int id)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

            var stockModel = await \_stockRepo.DeleteAsync(id);

            if (stockModel is null)

                return NotFound();

            return NoContent();

        }

    }

}

Шаг 8: Сервис для взаимодействия с внешним API

Для интеграции с внешним API (Financial Modeling Prep) был создан сервис, который выполняет HTTP-запросы с использованием бесплатного ключа для получения финансовых данных.

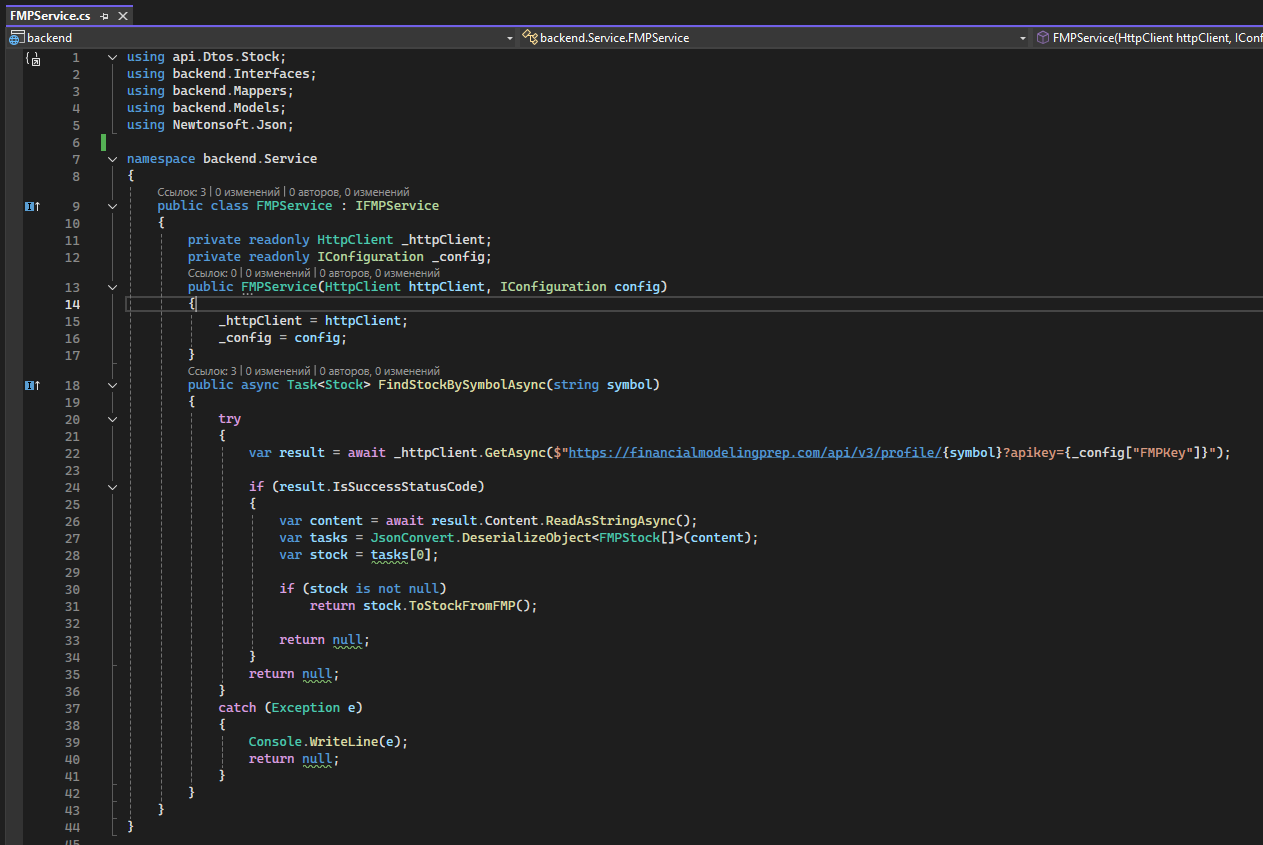


Рисунок 5. Листинг скрипта FMPService.cs.

Шаг 9: Тестирование с использованием Scalar

Для тестирования API был использован инструмент Scalar, который позволяет выполнять запросы к API, а также управлять заголовками и параметрами запросов.

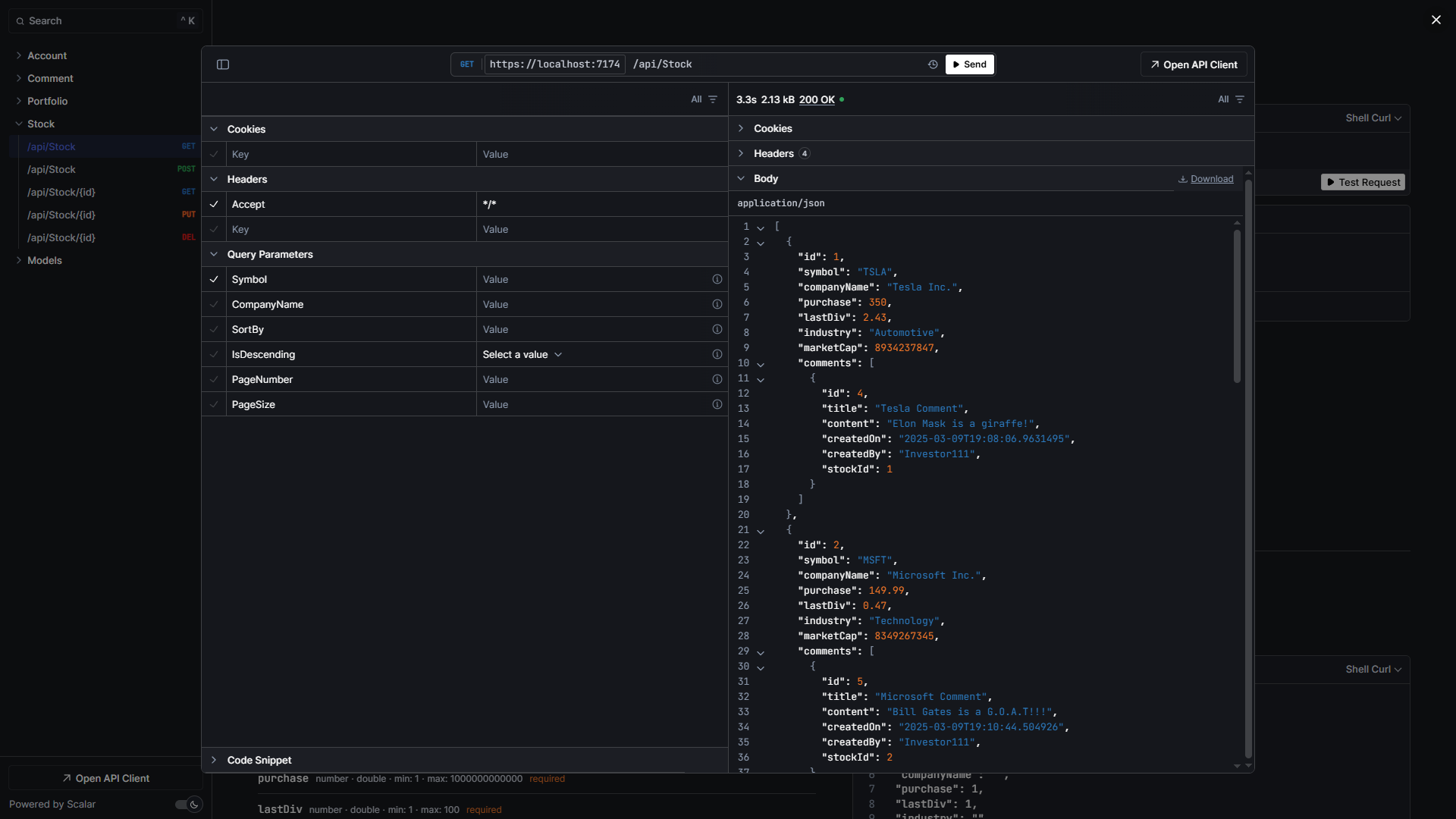


Рисунок 6. Пример результата GET-запроса в Scalar.

Список источников

1) YouTube. Плейлист "Backend-разработка API на ASP.NET Core". URL: <https://www.youtube.com/playlist?list=PL82C6-O4XrHcNJd4ejg8pX5fZaIDZmXyn> (дата обращения: 09.03.2025).

2) Кара, К. "Implementing the Repository Pattern in C# and .NET". Medium. URL: <https://medium.com/@kerimkkara/implementing-the-repository-pattern-in-c-and-net-5fdd91950485> (дата обращения: 09.03.2025).