ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ТОМСКОЙ ОБЛАСТИ ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 «Информационные системы и программирование»

«Разработка информационной системы проката инвентаря»

Пояснительная записка

к курсовому проекту

КП.25.09.02.07.631.11.ПЗ

Студент

«\_\_» \_\_\_\_\_\_\_\_ 20\_\_г. \_\_\_\_\_\_\_\_\_\_\_Э.А. Литвинчук

Руководитель

«\_\_» \_\_\_\_\_\_\_\_ 20\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_Д.Д. Демидов

Томск 2025г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc198263561)

[1. ОБЩАЯ ЧАСТЬ 4](#_Toc198263562)

[1.1. Описание предметной области 4](#_Toc198263563)

[1.2. Средства и среда разработки 4](#_Toc198263564)

[1.2.1 Выбор языка программирования 4](#_Toc198263565)

[1.2.2 Выбор СУБД 5](#_Toc198263566)

[1.2.3 Выбор сред разработки 6](#_Toc198263567)

[2. СПЕЦИАЛЬНАЯ ЧАСТЬ 7](#_Toc198263568)

[2.1. Описание функциональных требований 7](#_Toc198263569)

[2.2. Схема базы данных 9](#_Toc198263570)

[ЗАКЛЮЧЕНИЕ 10](#_Toc198263571)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 11](#_Toc198263572)

[ПРИЛОЖЕНИЕ А 12](#_Toc198263573)

[ПРИЛОЖЕНИЕ Б 38](#_Toc198263574)

# ВВЕДЕНИЕ

Современные тенденции развития цифровых технологий активно трансформируют сферу услуг, делая ее более удобной, автоматизированной и доступной для пользователей. Одним из востребованных направлений является прокат различного инвентаря – от спортивного оборудования до специализированных инструментов. Однако многие компании, занимающиеся арендой, до сих пор используют устаревшие методы учета и управления процессами, что приводит к потерям времени, ошибкам и неудовлетворенности клиентов.

Разработка информационной системы для проката инвентаря позволяет оптимизировать основные бизнес-процессы: учет оборудования, обработку заказов, контроль сроков аренды, расчет стоимости и взаимодействие с клиентами. Автоматизация этих процессов способствует повышению эффективности работы компании, снижению затрат и улучшению качества обслуживания.

Актуальность данной работы обусловлена необходимостью внедрения современных IT-решений в сферу проката для повышения конкурентоспособности бизнеса и удобства пользователей.

Целью курсового проекта является разработка информационной системы проката инвентаря, включающей функционал для учета оборудования, оформления аренды, управления клиентской базой и формирования отчетности.

Для достижения цели необходимо решить следующие задачи:

1. провести анализ предметной области;
2. определить требования к функционалу системы;
3. провести анализ и выбор средств разработки;
4. разработать архитектуру базы данных
5. разработать серверную часть системы;
6. протестировать разработанную систему.

# ОБЩАЯ ЧАСТЬ

## Описание предметной области

Система предназначена для автоматизации управления пунктом проката спортивного инвентаря. Она охватывает процессы учета инвентаря, взаимодействия с клиентами, оформления проката, расчета стоимости и формирования отчетности.

Функции, которые должна предоставлять система:

1. управление каталогом инвентаря;
2. управление данными о клиентах;
3. оформление проката;
4. расчёт стоимости проката;
5. отслеживание состояния инвентаря;
6. составление отчётов.

## Средства и среда разработки

Для разрабатываемой системы необходимо провести анализ и сделать выбор сред разработки, языка программирования и СУБД.

### Выбор языка программирования

Разрабатываемая система является клиент-серверное приложением, что подразумевает использования одного или нескольких языков программирования.

При выборе языка программирования для серверной части системы конкурировали два популярных языка программирования: «C#» и «PHP».

«C#» - это язык высокого уровня, разработанный компанией Microsoft, который предлагает высокую производительность, имеющий отличный потенциал для использования больших проектов ввиду хорошей масштабируемости, а также имеющий обширную документацию и большое сообщество.

«PHP» - это скриптовый язык, широко используемый для веб-разработки, который хорошо масштабируется, имеет очень большое сообщество и свободное распространение. Но данный язык проигрывает вышеописанному в производительности и удобстве масштабируемости.

В ходе анализа, предпочтение было отдано языку программирования «C#» и его фреймворку «ASP.NET Core», ввиду лучшей производительности, идеальной работы с потоками, поддержкой от Microsoft и удобной масштабируемостью.

При выборе языка программирования для клиентской части системы конкурировали два языка программирования: вышеописанный «C#» и «C++».

«C++» - это кроссплатформенный, объектно-ориентированный язык программирования, отличающийся высокой производительностью и популярностью, но требующий от разработчика высокой квалификации и трудозатрат для написания приложений.

В ходе анализа было выявлено, что «C#» более подходящий язык программирования для данного проекта, ввиду невысоких требований к производительности и малому количеству времени на разработку.

### Выбор СУБД

При выборе СУБД конкурировали две популярные системы управления реляционными базами данных: «PostgreSQL» и «MySQL».

«PostgreSQL» - это свободная объектно-реляционная система управления базами данных, имеющая открытый исходный код, строгую поддержку ACID, оптимизированная для обеспечения высокой производительности, поддерживающая сложные запросы и операции.

«MySQL» - это объектно-реляционная система управления базами данных, принадлежащая компании Oracle. Данная СУБД имеет меньше встроенных функций, что почти не играет роли в данном проекте, однако имеет лицензирование.

Разрабатываемый проект не имеет особых требований к СУБД, поэтому выбор между вышеперечисленными системами мог бы ограничиться рамками привычности команды разработчиков, однако предпочтение было отдано «PostgreSQL», ввиду невозможности использования лицензионной версии «MySQL» на территории России.

### Выбор сред разработки

После выбора языка программирования ясно, что необходима среда разработки с функционалом, поддерживающим «C#» и платформу «.Net». Выбор стоял между двумя мощными средами разработки: «Rider» и «Visual Studio».

«Rider» - это кросс-платформенная IDE для .NET-разработчиков, принадлежащая компании «JetBrains», основанная на платформе IntelliJ и ReSharper, позволяющая не только разрабатывать код, но и работать с фреймворками и СУБД.

«Visual Studio» - это интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для создания приложений, сервисов и инструментов для различных платформ. Данная среда разработки поддерживает множество языков программирования, что делает его универсальным инструментом для разработчиков.

В ходе анализа сред разработки был выбран «Rider», ввиду высшей производительности и удобств, связанных с работой с фреймворками и базами данных.

# СПЕЦИАЛЬНАЯ ЧАСТЬ

## Описание функциональных требований

Серверная часть должна отвечать следующим функциональным требованиям:

1. логирование;
2. авторизация по средством JWT;
3. получение списка инвентаря должно происходить при отправке запроса [GET] /api/inventory с возможностью фильтрации по параметрам: typeId (идентификатор типа инвентаря, опционально), statusId (идентификатор статуса, опционально), page (номер страницы, по умолчанию 1), pageSize (количество элементов на странице, по умолчанию 10);
4. получение информации о конкретном инвентаре должно происходить при отправке запроса [GET] /api/inventory/{id}, где id - идентификатор инвентаря;
5. добавление нового инвентаря должно происходить при отправке запроса [POST] /api/inventory, в теле которого должно храниться: name (название инвентаря), typeId (идентификатор типа), statusId (идентификатор статуса), pricePerHour (цена за час аренды);
6. изменение статуса инвентаря должно происходить при отправке запроса [PATCH] /api/inventory/{id}/status, в теле которого должен быть указан новый statusId (идентификатор статуса);
7. регистрация нового клиента должна происходить при отправке запроса [POST] /api/clients, в теле которого должно храниться: firstName (имя), lastName (фамилия), phone (номер телефона в формате +7XXXXXXXXXX), email (электронная почта, опционально);
8. поиск клиента должен происходить при отправке запроса [GET] /api/clients/search с параметром phone (номер телефона);
9. создание заказа должно происходить при отправке запроса [POST] /api/orders, в теле которого должно храниться: clientId (идентификатор клиента), items (массив объектов с inventoryId - идентификатор инвентаря, и hours - количество часов аренды), depositId (идентификатор залога, опционально);
10. возврат инвентаря должен происходить при отправке запроса [PATCH] /api/orders/{orderId}/return, в теле которого должен быть указан inventoryId (идентификатор возвращаемого инвентаря);
11. получение отчета о доходах должно происходить при отправке запроса [GET] /api/reports/income с параметрами: startDate (дата начала периода), endDate (дата окончания периода), groupBy (группировка по дням/неделям/месяцам);
12. получение отчета о популярном инвентаре должно происходить при отправке запроса [GET] /api/reports/popular с параметрами: limit (количество позиций в рейтинге), period (период для анализа);
13. получение списка типов инвентаря должно происходить при отправке запроса [GET] /api/dictionaries/types;
14. получение списка статусов должно происходить при отправке запроса [GET] /api/dictionaries/statuses.

## Схема базы данных

Для представления структуры базы данных необходимо построить логическую модель базы данных (рисунок 2.2.1).

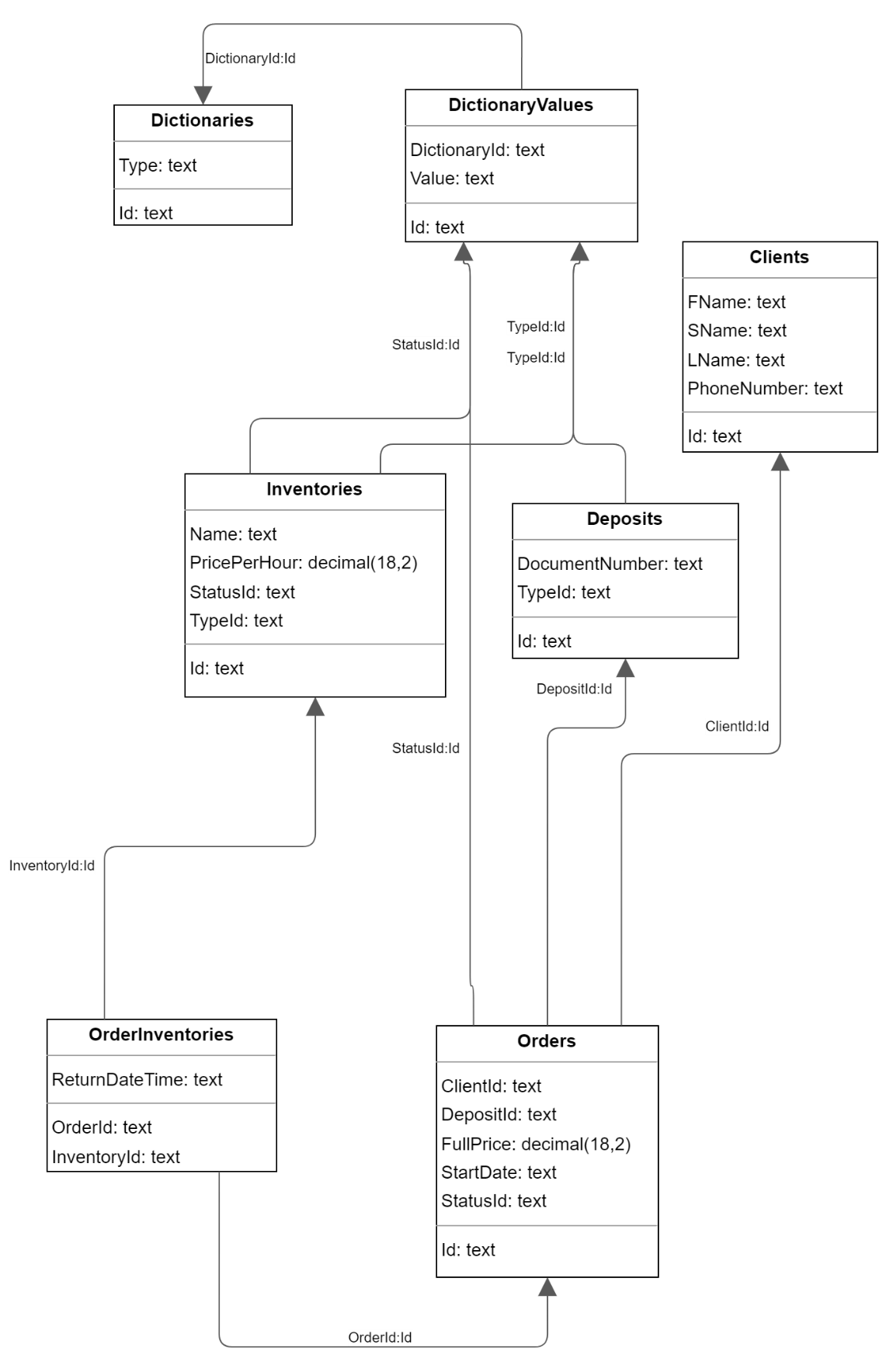


Рисунок 2.2.1 – Логическая схема данных

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта была разработана информационная система для учёта проката инвентаря. Система предоставляет API интерфейс для добавления и удаления клиентов, инвентаря и заказов, предоставляет возможности аутентификации клиента (например фронтенд-приложения), позволяет генерировать отчёты о заказах и инвентаре.

В рамках проекта был проведен анализ предметной области, анализ требований, построена диаграмма базы данных и определена функциональность системы, спроектирован API интерфейс.

Таким образом поставленная цель дипломного проекта была достигнута, в дальнейшем возможно изменение и дополнение системы при необходимости.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Литвиненко Иван. Руководство по использованию Avalonia UI в приложениях на C# / Литвиненко Иван. — [Онлайн]. — URL: https://avaloniaui.net/ (дата доступа: 08-02-2025).
2. Краткий обзор языка C# - Текст: Электронный // Microsoft - [сайт] <URL:https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/> Дата обращения: 5.02.2024.
3. Открытие общего диалогового окна (WPF .NET) Текст: Электронный / Microsoft - [сайт] URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/windows/how-to-open-common-system-dialog-box?view=netdesktop-8.0 Дата обращения: 08-02-2025.
4. Иэн Гриффитс. Программируем на С# 8.0 / Иэн Гриффитс – СПб: Питер, 2021. – 944 с. – Текст: непосредственный .
5. Джозеф Албахари, Бен Албахари. C# 9.0. Карманный справочник / Албахари Джозеф, Албахари Бен – СПб: Питер, 2021. – 256 c. - Текст : непосредственный.
6. Роб Майлз. The C# Programming Yellow Book / Роб Майлз – eBook, 2015. – 222 c. – Текст: непосредственный.
7. Статья «Основы программирования на C#» на портале C# Corner / Автор не указан. — [Онлайн]. — URL: https://www.c-sharpcorner.com/ (дата обращения: 08-02-2025).

# Листинг кода

1. AccuntController

using System.IdentityModel.Tokens.Jwt;

using AuditorGpt.Api.Data.Context;

using AuditorGpt.Api.Exceptions;

using AuditorGpt.Api.Interfases;

using AuditorGpt.Api.Models.DTOs;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

namespace AuditorGpt.Api.Controllers

{

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

[ApiController]

public class AccountController : ControllerBase

{

private readonly DataContext \_dataContext;

private readonly IPasswordHasher \_passwordHasher;

private readonly IJwtProvider \_jwtProvider;

public AccountController(DataContext dataContext, IPasswordHasher passwordHasher, IJwtProvider jwtProvider)

{

\_passwordHasher = passwordHasher;

\_dataContext = dataContext;

\_jwtProvider = jwtProvider;

}

[HttpPost]

public async Task<IActionResult> Login(LoginRequestDto loginRequest)

{

var user = await \_dataContext.Users

.Include(x => x.Role)

.FirstOrDefaultAsync(x => x.Login == loginRequest.Login);

if (user != null && !await \_passwordHasher.ValidateHashAsync(loginRequest.Password, user.PasswordHash))

throw new LoginException();

var token = await \_jwtProvider.GenerateJwtAsync(user);

var loginResponse = new LoginResponseDto

{

User = new UserDto

{

Id = user.Id,

Login = user.Login,

Password = user.PasswordHash,

FName = user.FName,

SName = user.SName,

LName = user.LName,

Photo = user.Photo,

RoleId = user.RoleId

},

Token = token

};

return Ok(loginResponse);

}

[Authorize]

[HttpPost("change-password")]

public async Task<IActionResult> ChangePassword(ChangePasswordRequestDto changePasswordRequest)

{

var user = await \_dataContext.Users

.Include(x => x.Role)

.FirstOrDefaultAsync(x => x.Login == changePasswordRequest.Login);

if (user != null && !await \_passwordHasher.ValidateHashAsync(

changePasswordRequest.OldPassword, user.PasswordHash))

throw new LoginException();

var hashedPassword = await \_passwordHasher.GenerateHashAsync(changePasswordRequest.NewPassword);

user.PasswordHash = hashedPassword;

\_dataContext.Users.Update(user);

await \_dataContext.SaveChangesAsync();

return Ok(user);

}

[HttpPost("load-photo")]

public async Task<IActionResult> AddPhotoProfileAsync(LoadPhotoDto loadPhoto)

{

var user = await \_dataContext.Users.FirstOrDefaultAsync(x => x.Id == loadPhoto.Id);

user.Photo = loadPhoto.Photo;

\_dataContext.Users.Update(user);

await \_dataContext.SaveChangesAsync();

return Ok(user.Photo);

}

}

}

1. ClientsController

using Core.DBModels;

using Data;

using Microsoft.AspNetCore.Mvc;

using RP.API.Dtos.Client;

using RP.API.Mapping;

using RP.Application.Interfaces;

namespace RP.API.Controllers;

[ApiController]

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

public class ClientsController : ControllerBase

{

private readonly IClientService \_clientService;

private readonly ClientsMapper \_mappingService;

public ClientsController(

IClientService clientService,

ClientsMapper mappingService)

{

\_clientService = clientService;

\_mappingService = mappingService;

}

/// <summary>

/// Создать нового клиента

/// </summary>

[HttpPost]

public async Task<ActionResult<ClientDto>> CreateClient([FromBody] ClientCreateDto dto)

{

var client = \_mappingService.MapToDomain(dto); // Явный вызов маппера

var createdClient = await \_clientService.CreateClientAsync(client);

var resultDto = \_mappingService.MapToDto(createdClient);

return CreatedAtAction(nameof(GetClient), new { id = resultDto.Id }, resultDto);

}

/// <summary>

/// Получить клиента по ID

/// </summary>

[HttpGet("{id}")]

public async Task<ActionResult<ClientDto>> GetClient(string id)

{

var client = await \_clientService.GetClientByIdAsync(id);

if (client == null)

{

return NotFound();

}

var dto = \_mappingService.MapToDto(client);

return Ok(dto);

}

}

1. InventoryController

using Core.DBModels;  
using Data;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.EntityFrameworkCore;  
using RP.API.Dtos.Inventory;  
using RP.API.Dtos.Satatuses;  
  
namespace RP.API.Controllers;  
  
[ApiController]  
[Route("api/inventory")]  
public class InventoryController : ControllerBase  
{  
 private readonly DataContext \_context;  
  
 public InventoryController(DataContext context)  
 {  
 \_context = context;  
 }  
 [HttpGet]  
 public async Task<ActionResult<IEnumerable<Inventory>>> GetAll()  
 {  
 return await \_context.Inventories  
 .Include(i => i.Type)  
 .Include(i => i.Status)  
 .ToListAsync();  
 }  
 [HttpGet("{id}")]  
 public async Task<ActionResult<Inventory>> GetById(string id)  
 {  
 var inventory = await \_context.Inventories  
 .Include(i => i.Type)  
 .Include(i => i.Status)  
 .FirstOrDefaultAsync(i => i.Id == id);  
  
 return inventory == null ? NotFound() : Ok(inventory);  
 }  
 [HttpPost]  
 public async Task<ActionResult<Inventory>> Create([FromBody] InventoryCreateDto dto)  
 {  
 var inventory = new Inventory  
 {  
 Name = dto.Name,  
 PricePerHour = dto.PricePerHour,  
 TypeId = dto.TypeId,  
 StatusId = dto.StatusId  
 };  
  
 \_context.Inventories.Add(inventory);  
 await \_context.SaveChangesAsync();  
  
 return CreatedAtAction(nameof(GetById), new { id = inventory.Id }, inventory);  
 }  
 [HttpPatch("{id}/status")]  
 public async Task<IActionResult> UpdateStatus(string id, [FromBody] StatusUpdateDto dto)  
 {  
 var inventory = await \_context.Inventories.FindAsync(id);  
 if (inventory == null) return NotFound();  
  
 inventory.StatusId = dto.StatusId;  
 await \_context.SaveChangesAsync();  
  
 return NoContent();  
 }  
}

1. OrdersController

using Microsoft.AspNetCore.Mvc;

using RP.API.Dtos.Inventory;

using RP.API.Dtos.Order;

using RP.API.Mapping;

using RP.Application.Interfaces;

namespace RP.API.Controllers;

[ApiController]

[Route("api/orders")]

public class OrdersController : ControllerBase

{

private readonly IOrderService \_orderService;

private readonly OrderMapper \_mapper;

public OrdersController(

IOrderService orderService,

OrderMapper mapper)

{

\_orderService = orderService;

\_mapper = mapper;

}

[HttpPost]

public async Task<ActionResult<OrderResponseDto>> CreateOrder([FromBody] OrderCreateDto dto)

{

var order = \_mapper.MapToDomain(dto);

var createdOrder = await \_orderService.CreateOrderAsync(order);

var resultDto = \_mapper.MapToDto(createdOrder);

return CreatedAtAction(nameof(GetOrder), new { id = resultDto.Id }, resultDto);

}

[HttpPost("{orderId}/return")]

public async Task<IActionResult> ReturnInventory(string orderId, [FromBody] ReturnInventoryDto dto)

{

await \_orderService.ReturnInventoryAsync(orderId, dto.InventoryId);

return NoContent();

}

/// <summary>

/// Получить заказ по ID

/// </summary>

/// <param name="id">Идентификатор заказа</param>

/// <response code="200">Заказ найден</response>

/// <response code="404">Заказ не найден</response>

[HttpGet("{id}")]

public async Task<ActionResult<OrderResponseDto>> GetOrder(string id)

{

var order = await \_orderService.GetOrderByIdAsync(id);

if (order == null)

{

return NotFound();

}

var dto = \_mapper.MapToDto(order);

return Ok(dto);

}

}

1. Program.cs

using RP.Application.Extentions;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddInfrastructure(builder.Configuration);

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseRouting();

app.UseHttpsRedirection();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

app.Run();

1. ClientService

using Core.DBModels;

using Data;

using RP.Application.Interfaces;

namespace RP.Application.Services;

/// <summary>

/// Сервис для работы с клиентами (работает только с доменными моделями)

/// </summary>

public class ClientService : IClientService

{

private readonly DataContext \_context;

public ClientService(DataContext context)

{

\_context = context;

}

/// <summary>

/// Создать нового клиента

/// </summary>

/// <param name="client">Доменная модель клиента</param>

/// <returns>Созданный клиент</returns>

public async Task<Client> CreateClientAsync(Client client)

{

\_context.Clients.Add(client);

await \_context.SaveChangesAsync();

return client;

}

/// <summary>

/// Получить клиента по идентификатору

/// </summary>

public async Task<Client?> GetClientByIdAsync(string id)

{

return await \_context.Clients.FindAsync(id);

}

}

1. InventoryService

using Core.DBModels;

using Data;

using Microsoft.EntityFrameworkCore;

using RP.Application.Interfaces;

namespace RP.Application.Services;

public class InventoryService : IInventoryService

{

private readonly DataContext \_context;

public InventoryService(DataContext context)

{

\_context = context;

}

/// <summary>

/// Получить весь инвентарь

/// </summary>

public async Task<IEnumerable<Inventory>> GetAllInventoriesAsync()

{

return await \_context.Inventories

.Include(i => i.Type)

.Include(i => i.Status)

.ToListAsync();

}

/// <summary>

/// Создать новый инвентарь

/// </summary>

/// <param name="dto">DTO с данными инвентаря</param>

public async Task<Inventory> CreateInventoryAsync(Inventory dto)

{

var inventory = new Inventory

{

Name = dto.Name,

PricePerHour = dto.PricePerHour,

TypeId = dto.TypeId,

StatusId = dto.StatusId

};

\_context.Inventories.Add(inventory);

await \_context.SaveChangesAsync();

return inventory;

}

}

1. OrderService

using Core.DBModels;

using Data;

using Microsoft.EntityFrameworkCore;

using RP.Application.Interfaces;

namespace RP.Application.Services;

public class OrderService : IOrderService

{

private readonly DataContext \_context;

public OrderService(DataContext context)

{

\_context = context;

}

public async Task<Order> CreateOrderAsync(Order order)

{

\_context.Orders.Add(order);

await \_context.SaveChangesAsync();

return order;

}

public Task<Order?> GetOrderByIdAsync(string id)

{

throw new NotImplementedException();

}

public async Task ReturnInventoryAsync(string orderId, string inventoryId)

{

var orderItem = await \_context.OrderInventories

.FirstOrDefaultAsync(oi => oi.OrderId == orderId && oi.InventoryId == inventoryId);

if (orderItem != null)

{

orderItem.ReturnDateTime = DateTime.UtcNow;

await \_context.SaveChangesAsync();

}

}

}

1. BaseIdEntity

namespace Core.DBModels;

/// <summary>

/// Базовая сущность, содержащая идентификатор.

/// </summary>

public abstract class BaseIdEntity

{

/// <summary>

/// Идентификатор сущности.

/// </summary>

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

}

1. Client

namespace Core.DBModels;

/// <summary>

/// Клиент.

/// </summary>

public class Client : BaseIdEntity

{

/// <summary>

/// Имя.

/// </summary>

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

/// <summary>

/// Фамилия.

/// </summary>

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

/// <summary>

/// Отчество.

/// </summary>

public string? LName { get; set; }

/// <summary>

/// Номер телефона.

/// </summary>

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

/// <summary>

/// Навигационное св-во заказов.

/// </summary>

public ICollection<Order>? OrderLinks { get; set; }

}

1. Deposit

namespace Core.DBModels;

/// <summary>

/// Залог.

/// </summary>

public class Deposit : BaseIdEntity

{

/// <summary>

/// Номер документа.

/// </summary>

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

/// <summary>

/// Тип документа.

/// </summary>

public DictionaryValue Type { get; set; } = null!;

public string TypeId { get; set; }

/// <summary>

/// Навигационное св-во заказов.

/// </summary>

public ICollection<Order>? OrderLinks { get; set; }

}

1. Dictionary

using System.Collections.ObjectModel;

using System.ComponentModel.DataAnnotations;

namespace Core.DBModels;

/// <summary>

/// Тип справочника.

/// </summary>

public class Dictionary : BaseIdEntity

{

/// <summary>

/// Значения справочника данного типа.

/// </summary>

public ICollection<DictionaryValue?> DictionaryValues { get; set; } = new Collection<DictionaryValue?>();

/// <summary>

/// Наименование типа справочника.

/// </summary>

[Required]

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

}

1. DictionaryValue

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Core.DBModels;

/// <summary>

/// Значение справочника.

/// </summary>

public class DictionaryValue : BaseIdEntity

{

/// <summary>

/// Тип справочника.

/// </summary>

[Required]

public Dictionary Dictionary { get; set; } = null!;

public string DictionaryId { get; set; }

/// <summary>

/// Значение справочника.

/// </summary>

[Required]

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

}

1. Inventory

namespace Core.DBModels;

/// <summary>

/// Инвентарь.

/// </summary>

public class Inventory : BaseIdEntity

{

/// <summary>

/// Название.

/// </summary>

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

/// <summary>

/// Стоимость за час.

/// </summary>

public decimal PricePerHour { get; set; }

/// <summary>

/// Тип инвентаря.

/// </summary>

public DictionaryValue Type { get; set; } = null!;

/// <summary>

/// Идентификатор типа.

/// </summary>

public string TypeId { get; set; }

/// <summary>

/// Статус.

/// </summary>

public DictionaryValue Status { get; set; } = null!;

/// <summary>

/// Идентификатор статуса.

/// </summary>

public string StatusId { get; set; }

/// <summary>

/// Коллекция связей с заказом.

/// </summary>

public ICollection<OrderInventories> OrderInventories { get; set; } = new List<OrderInventories>();

}

1. Order

namespace Core.DBModels;

/// <summary>

/// Заказ.

/// </summary>

public class Order : BaseIdEntity

{

/// <summary>

/// Начало оформления заказа.

/// </summary>

public DateTime StartDate { get; set; }

/// <summary>

/// Полная стоимость.

/// </summary>

public decimal? FullPrice { get; set; }

/// <summary>

/// Клиент.

/// </summary>

public Client? Client { get; set; }

public string? ClientId { get; set; }

/// <summary>

/// Статус.

/// </summary>

public DictionaryValue Status { get; set; } = null!;

/// <summary>

/// Идентификатор статуса.

/// </summary>

public string StatusId { get; set; }

/// <summary>

/// Залог.

/// </summary>

public Deposit? Deposit { get; set; }

/// <summary>

/// Идентификатор залога.

/// </summary>

public string? DepositId { get; set; }

/// <summary>

/// Коллекция связей с инвентарём.

/// </summary>

public ICollection<OrderInventories> OrderInventories { get; set; } = new List<OrderInventories>();

}

1. OrderInventories

namespace Core.DBModels;

/// <summary>

/// Связующая таблица заказа и инвентаря.

/// </summary>

public class OrderInventories

{

/// <summary>

/// Идентификатор заказа.

/// </summary>

public string OrderId { get; set; }

/// <summary>

/// Навигационное св-во заказа.

/// </summary>

public Order Order { get; set; }

/// <summary>

/// Идентификатор инвентаря.

/// </summary>

public string InventoryId { get; set; }

/// <summary>

/// Навигационное св-во инвентаря.

/// </summary>

public Inventory Inventory { get; set; }

/// <summary>

/// Дата и время возврата инвентаря.

/// </summary>

public DateTime ReturnDateTime { get; set; }

}

1. DataContext

using Core.DBModels;

using Microsoft.EntityFrameworkCore;

namespace Data;

/// <summary>

/// Контекст БД.

/// </summary>

public class DataContext : DbContext

{

/// <summary>

/// Конструктор.

/// </summary>

public DataContext() { }

/// <summary>

/// Конструктор с опциями.

/// </summary>

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

/// <inheritdoc cref="Client"/>

public DbSet<Client> Clients { get; set; }

/// <inheritdoc cref="Deposit"/>

public DbSet<Deposit> Deposits { get; set; }

/// <inheritdoc cref="Dictionary"/>

public DbSet<Dictionary> Dictionaries { get; set; }

/// <inheritdoc cref="DictionaryValue"/>

public DbSet<DictionaryValue> DictionaryValues { get; set; }

/// <inheritdoc cref="Inventory"/>

public DbSet<Inventory> Inventories { get; set; }

/// <inheritdoc cref="Order"/>

public DbSet<Order> Orders { get; set; }

/// <inheritdoc cref="OrderInventories"/>

public DbSet<OrderInventories> OrderInventories { get; set; }

/// <summary>

/// Конфигурация контекста для работы с БД.

/// </summary>

/// <param name="optionsBuilder"> Опции контекста. </param>

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var currentDirectory = Directory.GetCurrentDirectory();

var basePath = currentDirectory.Replace(@"\RP.API\bin\Debug\net8.0", "");

var dbPath = Path.Combine(basePath, "db.sqlite");

optionsBuilder.UseSqlite($"Data Source={dbPath};");

}

/// <summary>

/// Конфигурация моделей.

/// </summary>

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Конфигурация Client

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.FName).IsRequired().HasMaxLength(100);

entity.Property(e => e.SName).IsRequired().HasMaxLength(100);

entity.Property(e => e.PhoneNumber).IsRequired().HasMaxLength(20);

});

// Конфигурация Deposit

modelBuilder.Entity<Deposit>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.DocumentNumber).IsRequired().HasMaxLength(50);

entity.HasOne(d => d.Type)

.WithMany()

.HasForeignKey(d => d.TypeId)

.OnDelete(DeleteBehavior.Restrict);

});

// Конфигурация Dictionary

modelBuilder.Entity<Dictionary>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Type).IsRequired().HasMaxLength(100);

entity.HasIndex(e => e.Type).IsUnique();

});

// Конфигурация DictionaryValue

modelBuilder.Entity<DictionaryValue>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Value).IsRequired().HasMaxLength(200);

entity.HasOne(dv => dv.Dictionary)

.WithMany(d => d.DictionaryValues)

.HasForeignKey(dv => dv.DictionaryId)

.OnDelete(DeleteBehavior.Cascade);

});

// Конфигурация Inventory

modelBuilder.Entity<Inventory>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Name).IsRequired().HasMaxLength(200);

entity.Property(e => e.PricePerHour).HasColumnType("decimal(18,2)");

entity.HasOne(i => i.Type)

.WithMany()

.HasForeignKey(i => i.TypeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(i => i.Status)

.WithMany()

.HasForeignKey(i => i.StatusId)

.OnDelete(DeleteBehavior.Restrict);

});

// Конфигурация Order

modelBuilder.Entity<Order>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.StartDate).IsRequired();

entity.Property(e => e.FullPrice).HasColumnType("decimal(18,2)");

entity.HasOne(o => o.Client)

.WithMany(c => c.OrderLinks)

.HasForeignKey(o => o.ClientId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasOne(o => o.Status)

.WithMany()

.HasForeignKey(o => o.StatusId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(o => o.Deposit)

.WithMany(d => d.OrderLinks)

.HasForeignKey(o => o.DepositId)

.OnDelete(DeleteBehavior.SetNull);

});

// Конфигурация OrderInventories (новая связующая таблица)

modelBuilder.Entity<OrderInventories>(entity =>

{

entity.HasKey(oi => new { oi.OrderId, oi.InventoryId });

entity.Property(oi => oi.ReturnDateTime).IsRequired();

entity.HasOne(oi => oi.Order)

.WithMany(o => o.OrderInventories)

.HasForeignKey(oi => oi.OrderId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(oi => oi.Inventory)

.WithMany(i => i.OrderInventories)

.HasForeignKey(oi => oi.InventoryId)

.OnDelete(DeleteBehavior.Restrict);

});

}

}

# Результаты работы программы

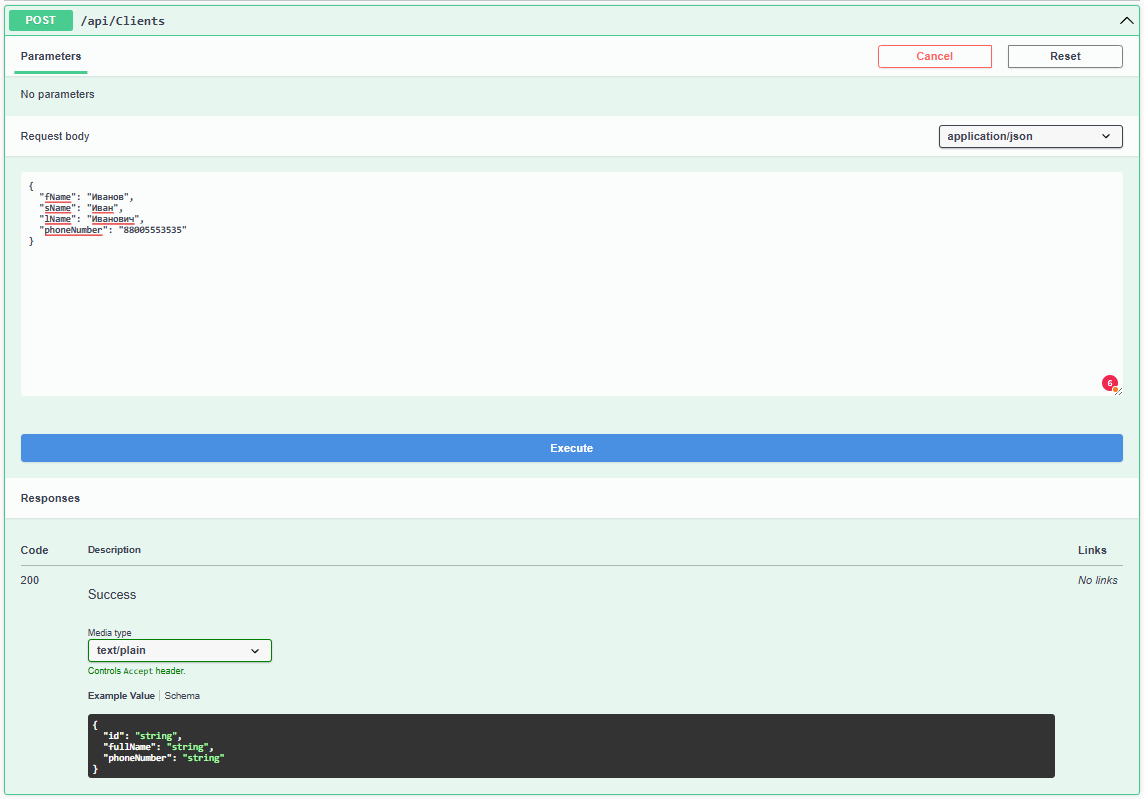


Рисунок Б.1 – Добавление пользователя в систему.

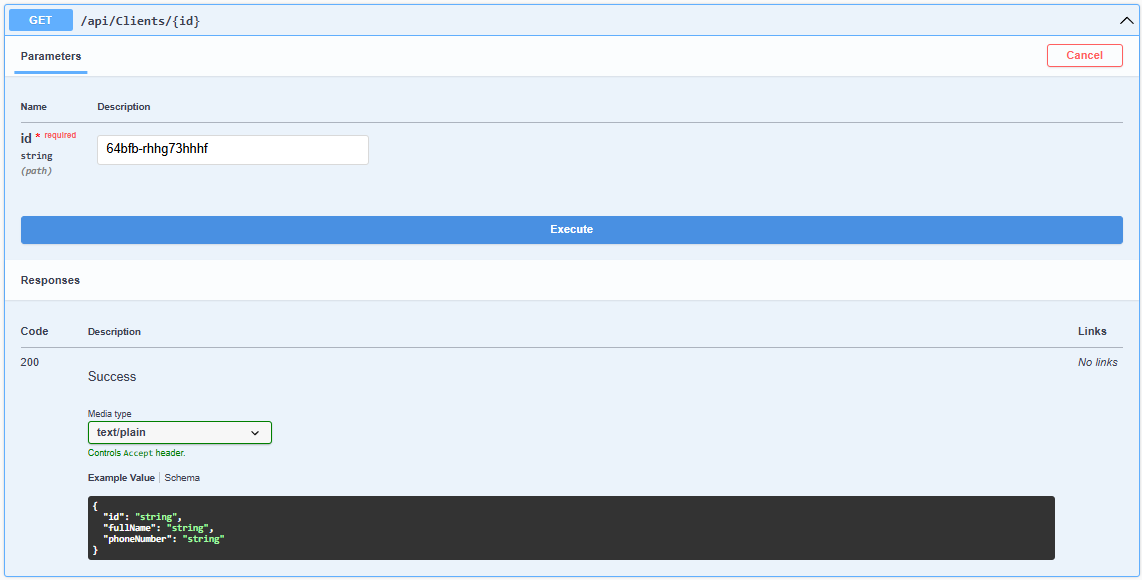


Рисунок Б.2 – Получение клиента по идентификатору.

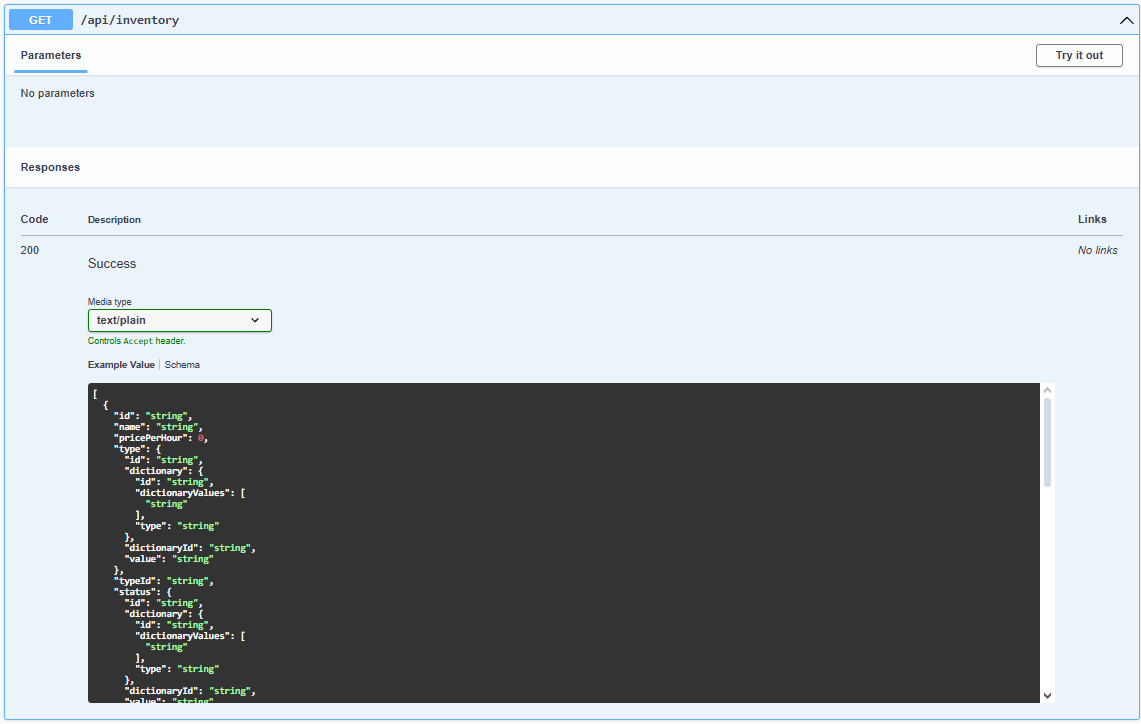


Рисунок Б.3 - Получение всего инвентаря.

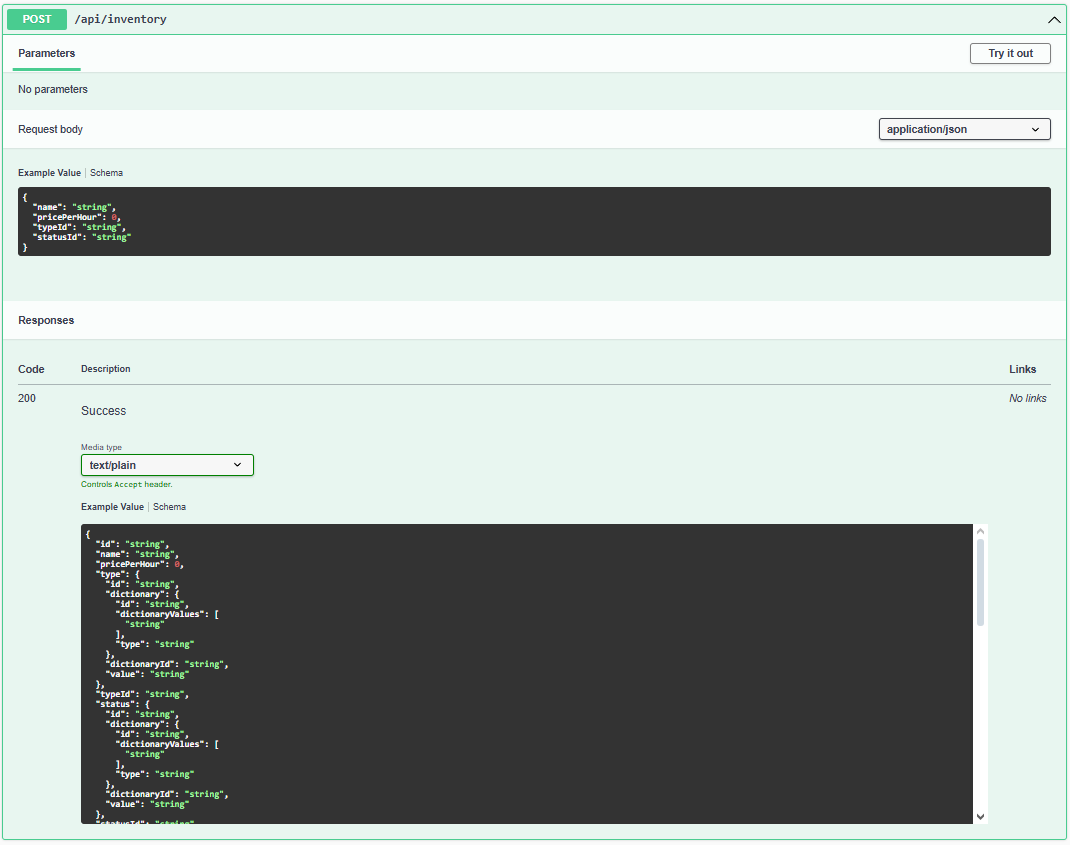


Рисунок Б.4 - Создание инвентаря в системе.

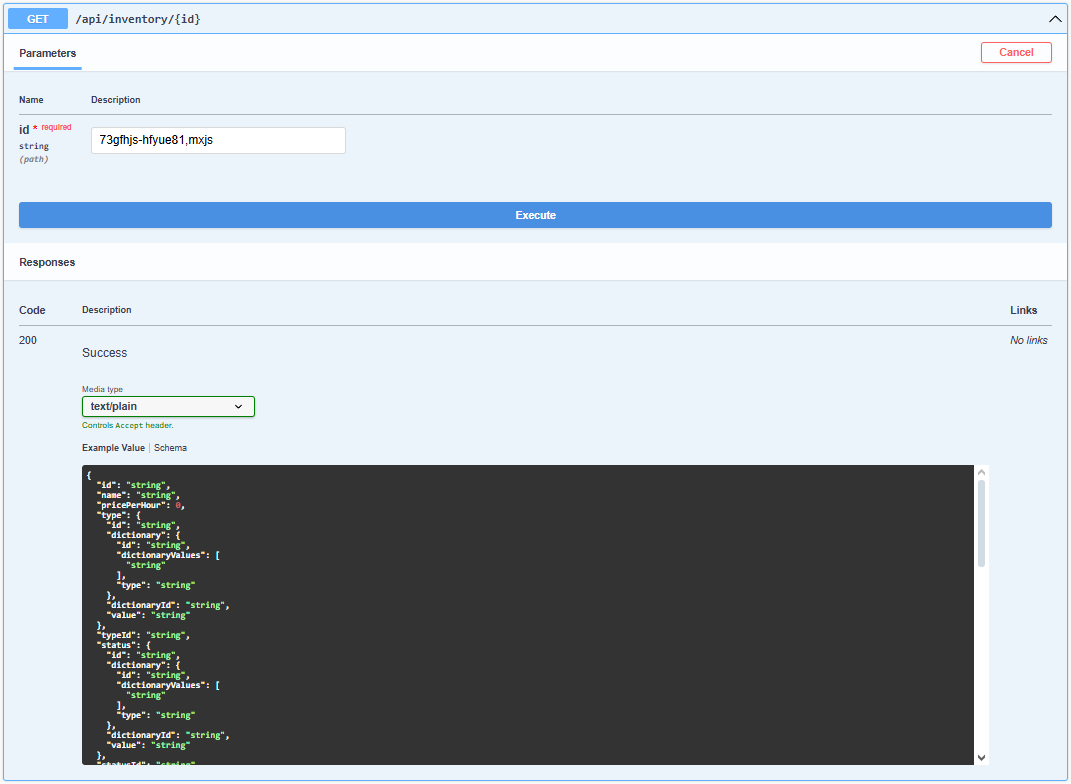


Рисунок Б.5 - Получение инвентаря по идентификатору.

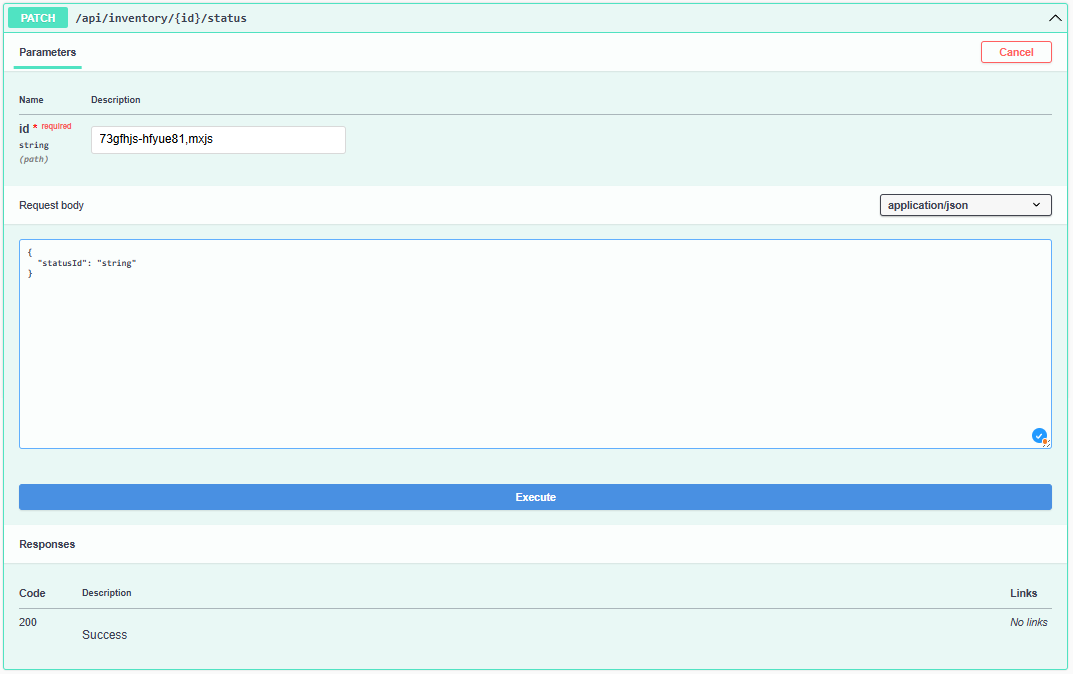


Рисунок Б.6 - Изменение статуса заказа.

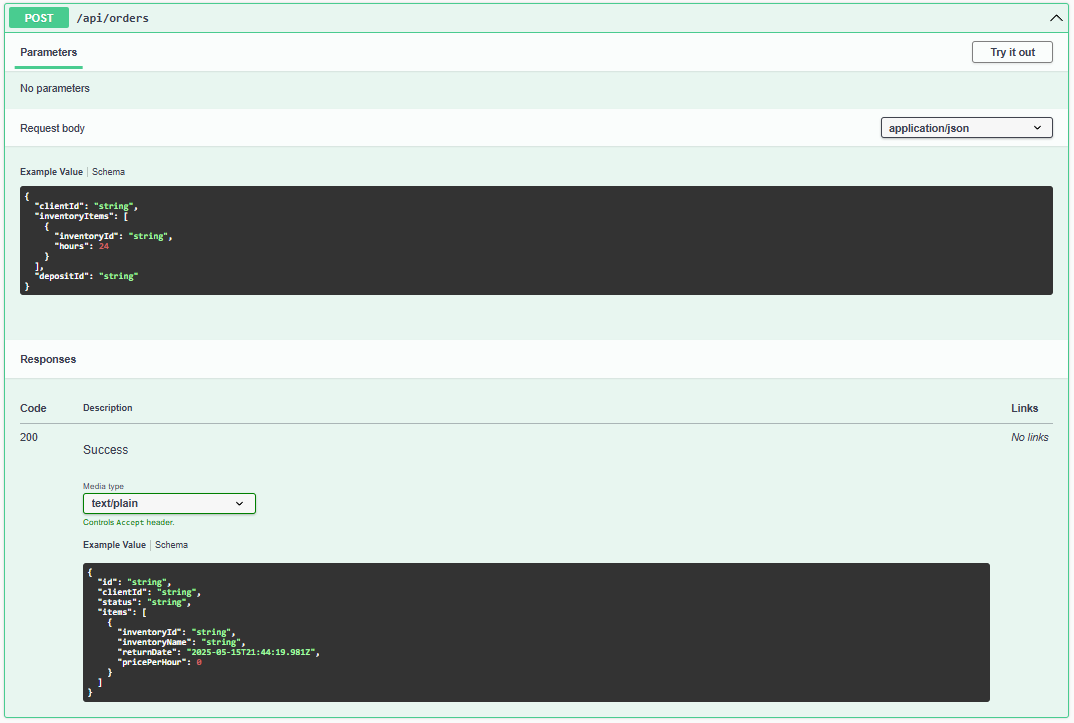


Рисунок Б.7 - Получение всех заказов.

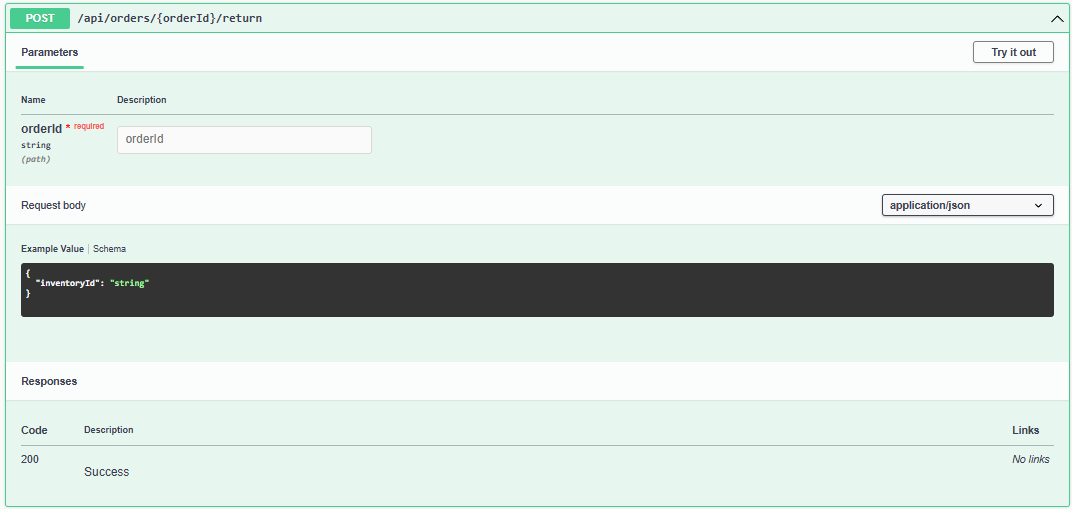


Рисунок Б.8 - Возврат инвентаря в заказе.

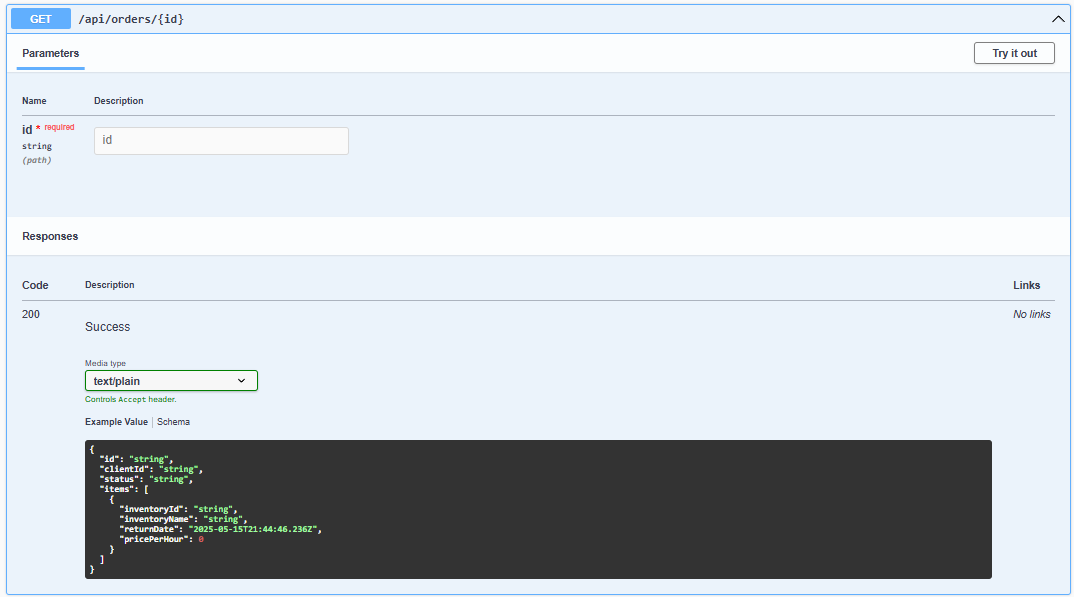


Рисунок Б.9 - Получение заказа по идентификатору.

Результат работы авторизации (рисунок Б.3 - рисунок Б.5).

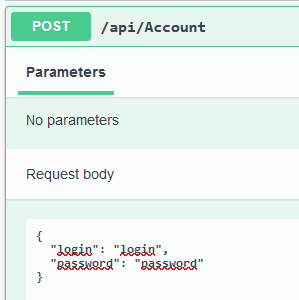


Рисунок Б.10 - Результат работы авторизации

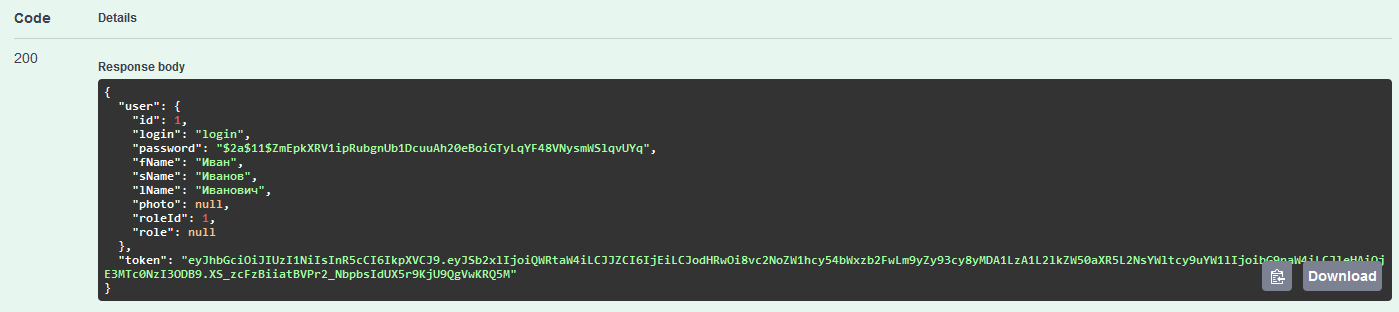


Рисунок Б.11 - Результат работы авторизации

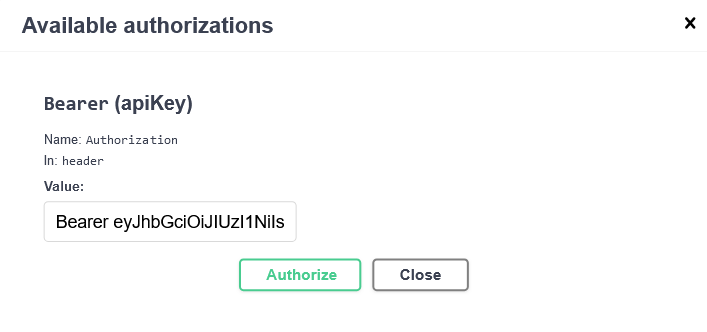


Рисунок Б.12 - Результат работы авторизации