ДЕПАРТАМЕНТ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ТОМСКОЙ ОБЛАСТИ

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 «Информационные системы и программирование»

Разработка информационно-справочной системы для автоматизации расписания железнодорожной станции.

Пояснительная записка

к курсовому проекту

КП.23.090207.621.12.ПЗ

|  |  |
| --- | --- |
| Студент |  |
| «\_\_»\_\_\_\_\_\_\_ 2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_ Д. А. Файт |
| Руководитель |  |
| «\_\_»\_\_\_\_\_\_\_ 2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_ А. Ю. Маюнова |

Томск 2024

СОДЕРЖАНИЕ

[1 ОБЩАЯ ЧАСТЬ 3](#_Toc169740585)

[1.1 Анализ предметной области 3](#_Toc169740586)

[1.2 Выбор сред и средств разработки 3](#_Toc169740587)

[2 СПЕЦИАЛЬНАЯ ЧАСТЬ 6](#_Toc169740588)

[2.1 Описание требований к информационной системе 6](#_Toc169740589)

[2.2 Диаграмма вариантов использования 6](#_Toc169740590)

[2.3 Схема данных 6](#_Toc169740591)

[2.4 Пользовательские сценарии 6](#_Toc169740592)

[3 ЗАКЛЮЧЕНИЕ 7](#_Toc169740593)

[ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 8](#_Toc169740594)

[Листинг программы 9](#_Toc169740595)

[Результаты работы программы 74](#_Toc169740596)

# ОБЩАЯ ЧАСТЬ

## Анализ предметной области

Анализ предметной области для информационно-справочной системы, предназначенной для автоматизации расписания железнодорожной станции, включает в себя определение основных функций, участников, данных и процессов, которые должны быть учтены при разработке системы. Вот детальный анализ:

**1. Цели и задачи системы**

* **Цель**: Улучшение эффективности управления расписанием поездов на железнодорожной станции.
* **Задачи**:
  + Автоматизация составления и корректировки расписания.
  + Обеспечение доступа к актуальной информации о поездах для персонала станции и пассажиров.
  + Управление данными о путевых и технических возможностях станции.
  + Интеграция с системами других станций и диспетчерских центров.

**2. Основные участники и их роли**

* **Диспетчеры**: Составление и корректировка расписания, управление движением поездов.
* **Персонал станции**: Обеспечение соблюдения расписания, техническое обслуживание.
* **Пассажиры**: Получение информации о расписании и изменениях.
* **Администрация железной дороги**: Управление и контроль за функционированием системы.

**3. Основные данные и информационные потоки**

* **Расписание поездов**: Время прибытия и отправления, маршруты, типы поездов.
* **Данные о путевых и технических возможностях**: Состояние путей, наличие технических средств, возможности по приему и отправлению поездов.
* **Информация о пассажирах**: Данные о бронировании и билетах (при наличии интеграции с системами бронирования).
* **Сигналы и сообщения**: Информация о задержках, изменениях маршрутов, аварийных ситуациях.

**4. Основные процессы**

* **Составление расписания**: Ввод данных о поездах, определение оптимальных маршрутов и временных интервалов.
* **Корректировка расписания**: Реагирование на изменения (погодные условия, технические проблемы, аварии).
* **Информирование участников**: Оповещение персонала и пассажиров о изменениях в расписании.
* **Контроль и мониторинг**: Наблюдение за соблюдением расписания, анализ эффективности работы станции.

**5. Технические требования**

* **Интерфейс**: Удобный и интуитивно понятный для всех категорий пользователей.
* **Безопасность данных**: Защита конфиденциальной информации, резервное копирование данных.
* **Масштабируемость**: Возможность расширения системы при развитии железнодорожной сети.
* **Интеграция**: Совместимость с существующими системами управления железнодорожным транспортом.

**6. Оценка рисков**

* **Технические сбои**: Недоступность системы из-за технических проблем.
* **Ошибки в данных**: Некорректное составление или корректировка расписания.
* **Кибербезопасность**: Угрозы со стороны хакеров или вирусных атак.

**7. Этапы разработки и внедрения**

* **Анализ и проектирование**: Определение требований, проектирование архитектуры системы.
* **Разработка**: Создание программного обеспечения, тестирование.
* **Внедрение**: Установка системы на железнодорожной станции, обучение персонала.
* **Сопровождение и обновления**: Поддержание системы в рабочем состоянии, внесение изменений и улучшений.

Этот анализ предметной области служит основой для разработки информационно-справочной системы, которая будет эффективно поддерживать управление расписанием на железнодорожной станции.

## Выбор сред и средств разработки

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

1. Выбор языка программирования

При выборе языка программирования для сервера 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.

1. 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.

1. 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 требований к информационной системе

Для разработки проекта необходимо предъявить требования к его функциональности и технологиям.

1. Требования к функциональности
2. Требования

## Диаграмма вариантов использования

## Схема данных

Для представления структуры базы данных необходимо построить логическую модель базы данных (рисунок 2.4.1).

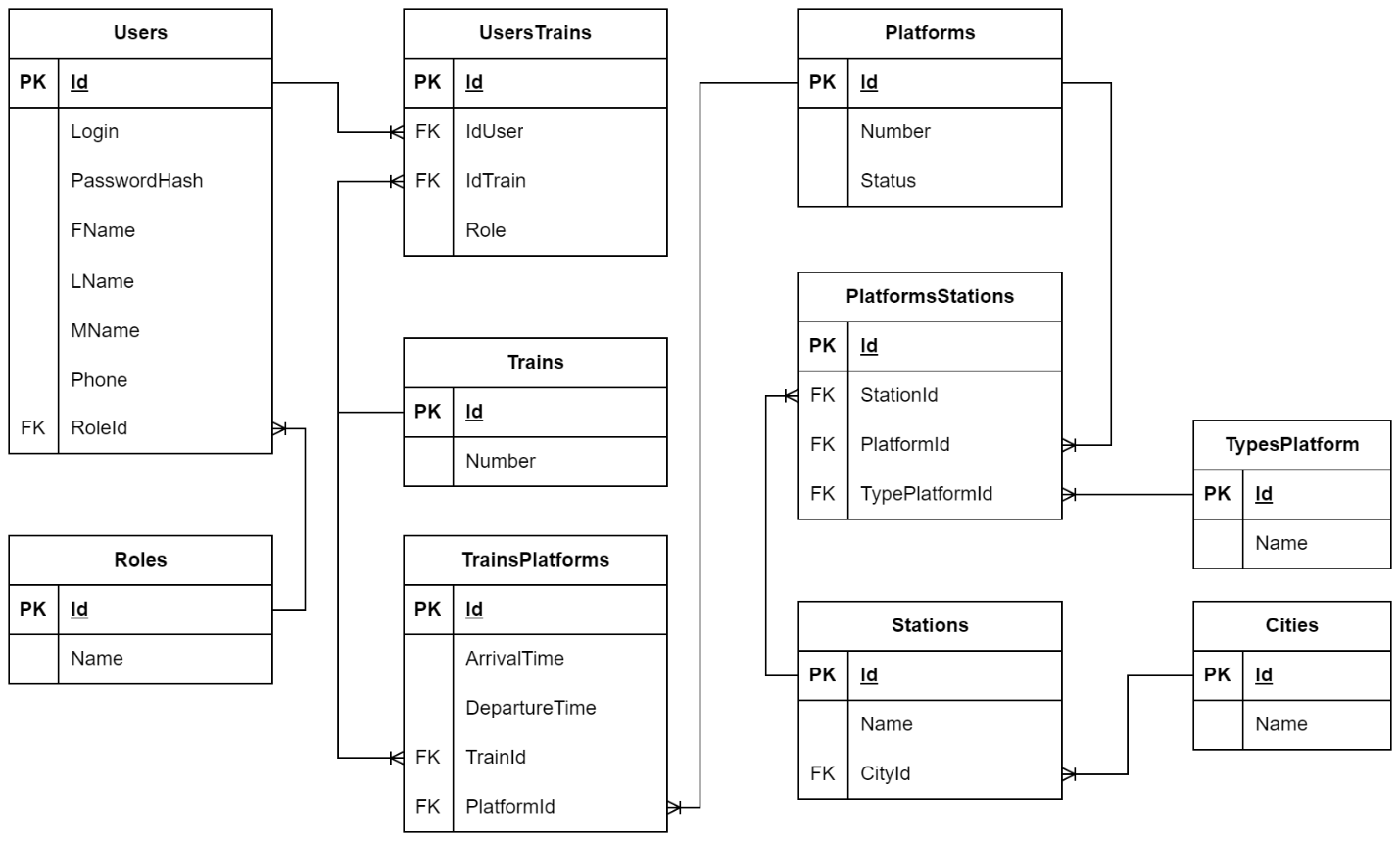


Рисунок 2.3.1 - Схема данных (логическая модель базы данных)

## Пользовательские сценарии

# ЗАКЛЮЧЕНИЕ

# ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. 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.
2. Иэн Гриффитс. Программируем на С# 8.0 / Иэн Гриффитс – СПб: Питер, 2021. – 944 с. – Текст: непосредственный .
3. Джозеф Албахари, Бен Албахари. C# 9.0. Карманный справочник / Албахари Джозеф, Албахари Бен – СПб: Питер, 2021. – 256 c. - Текст : непосредственный.
4. Роб Майлз. The C# Programming Yellow Book / Роб Майлз – eBook, 2015. – 222 c. – Текст: непосредственный.
5. Статья «Основы программирования на C#» на портале C# Corner / Автор не указан. — [Онлайн]. — URL: https://www.c-sharpcorner.com/ (дата обращения: 30-01-2024).
6. Статья «Программирование на C#» на сайте Microsoft Docs / Microsoft Corporation. — [Онлайн]. — URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата доступа: 27-04-2024).

Книга «Разработка приложений на C# с использованием Avalonia UI» на сайте издательства «Лори» / Автор не указан. — [Онлайн]. — URL: https://lori.ru/ (дата обращения: 19-05-2024).

# Листинг программы

1. AccountController.cs

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using TrainTimings.Api.DTOs.Account;

using TrainTimings.Application.Interfaces.IServices;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class AccountController : ControllerBase

{

private readonly IAccountService \_accountService;

public AccountController(IAccountService accountService)

{

\_accountService = accountService;

}

[HttpPost("login")]

public async Task<IActionResult> Login(LoginRequestDto loginRequestDto)

{

var result = await \_accountService.LoginAsync(loginRequestDto.Login, loginRequestDto.Password);

return Ok(result);

}

[HttpPost("change-password")]

public async Task<IActionResult> ChangePassword(ChangePasswordDto request)

{

await \_accountService.ChangePasswordAsync(request.Username, request.OldPassword, request.NewPassword);

return Ok();

}

}

}

1. CityController.cs

using AutoMapper;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using TrainTimings.Api.DTOs.City;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class CityController : ControllerBase

{

private readonly ICityService \_cityService;

private readonly IMapper \_mapper;

public CityController(ICityService cityService, IMapper mapper)

{

\_cityService = cityService;

\_mapper = mapper;

}

[HttpGet("get-all")]

public async Task<IActionResult> GetCities()

{

var result = await \_cityService.GetAllAsync();

return Ok(result);

}

[HttpGet("get-by-id/{id}")]

public async Task<IActionResult> GetCityById(int id)

{

var result = await \_cityService.GetByIdAsync(id);

return Ok(result);

}

[HttpPost("create")]

public async Task<IActionResult> CreateCity(CreateCityDto cityRequest)

{

var city = \_mapper.Map<City>(cityRequest);

var result = await \_cityService.CreateAsync(city);

return Ok(result);

}

[HttpPut("update")]

public async Task<IActionResult> UpdateCity(UpdateCityDto cityRequest)

{

var city = \_mapper.Map<City>(cityRequest);

var result = await \_cityService.UpdateAsync(city);

return Ok(result);

}

[HttpDelete("delete/{id}")]

public async Task<IActionResult> DeleteCity(int id)

{

await \_cityService.DeleteAsync(id);

return NoContent();

}

}

}

1. TimingControlle.cs

using AutoMapper;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using TrainTimings.Api.DTOs.Timing;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class TimingControlle : ControllerBase

{

private readonly ITimingService \_timingService;

private readonly IMapper \_mapper;

public TimingControlle(ITimingService timingService, IMapper mapper)

{

\_timingService = timingService;

\_mapper = mapper;

}

[HttpGet("get-all")]

public async Task<IActionResult> GetAll()

{

var timings = await \_timingService.GetAllTimingsAsync();

return Ok(timings);

}

[HttpGet("get-by-id/{id}")]

public async Task<IActionResult> GetById(int id)

{

var timing = await \_timingService.GetTimingByIdAsync(id);

return Ok(timing);

}

[HttpPost("create")]

public async Task<IActionResult> Create(CreateTimingDto timing)

{

var createdTiming = await \_timingService.CreateTimingAsync(\_mapper.Map<Timing>(timing));

return Ok(createdTiming);

}

[HttpPut("update")]

public async Task<IActionResult> Update(UpdateTimingDto timing)

{

var updatedTiming = await \_timingService.UpdateTimingAsync(\_mapper.Map<Timing>(timing));

return Ok(updatedTiming);

}

[HttpDelete("delete/{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_timingService.DeleteTimingAsync(id);

return NoContent();

}

}

}

1. TrainController.cs

using AutoMapper;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using TrainTimings.Api.DTOs.Train;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class TrainController : ControllerBase

{

private readonly ITrainService \_trainService;

private readonly IMapper \_mapper;

public TrainController(ITrainService trainService, IMapper mapper)

{

\_trainService = trainService;

\_mapper = mapper;

}

[HttpGet("get-all")]

public async Task<IActionResult> GetTrains()

{

var result = await \_trainService.GetAllAsync();

return Ok(result);

}

[HttpGet("get-by-id/{id}")]

public async Task<IActionResult> GetTrainById(int id)

{

var result = await \_trainService.GetByIdAsync(id);

return Ok(result);

}

[HttpGet("get-by-number/{number}")]

public async Task<IActionResult> GetTrainByNumber(string number)

{

var result = await \_trainService.GetByNumberAsync(number);

return Ok(result);

}

[HttpPost("create")]

public async Task<IActionResult> CreateTrain(CreateTrainDto trainRequest)

{

var result = await \_trainService.CreateAsync(\_mapper.Map<Train>(trainRequest));

return Ok(result);

}

[HttpPut("update")]

public async Task<IActionResult> UpdateTrain(UpdateTrainDto trainRequest)

{

var result = await \_trainService.UpdateAsync(\_mapper.Map<Train>(trainRequest));

return Ok(result);

}

[HttpDelete("delete/{id}")]

public async Task<IActionResult> DeleteTrain(int id)

{

await \_trainService.DeleteAsync(id);

return NoContent();

}

}

}

1. TypeFollowingController.cs

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class TypeFollowingController : ControllerBase

{

}

}

1. TypeTrainController.cs

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class TypeTrainController : ControllerBase

{

}

}

1. UserController.cs

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace TrainTimings.Api.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

}

}

1. MappingProfile.cs

using System.Reflection.PortableExecutable;

using AutoMapper;

using TrainTimings.Api.DTOs.City;

using TrainTimings.Api.DTOs.Timing;

using TrainTimings.Api.DTOs.Train;

using TrainTimings.Core.Models;

namespace TrainTimings.Api.Mapping;

public class MappingProfile : Profile

{

public MappingProfile()

{

CreateMap<CreateCityDto, City>().ReverseMap();

CreateMap<UpdateCityDto, City>().ReverseMap();

CreateMap<CreateTimingDto, Timing>().ReverseMap();

CreateMap<UpdateTimingDto, Timing>().ReverseMap();

CreateMap<CreateTrainDto, Train>().ReverseMap();

CreateMap<UpdateTimingDto, Train>().ReverseMap();

}

}

1. CustomExceptionHandlerMiddleware.cs

using System.ComponentModel.DataAnnotations;

using System.Net;

using System.Text.Json;

using TrainTimings.Application.Exceptions;

namespace TrainTimings.Api.Middlewares;

public class CustomExceptionHandlerMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<CustomExceptionHandlerMiddleware> \_logger;

public CustomExceptionHandlerMiddleware(RequestDelegate next, ILogger<CustomExceptionHandlerMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception e)

{

await HandleExceptionAsync(context, e);

}

}

private Task HandleExceptionAsync(HttpContext context, Exception exception)

{

var code = HttpStatusCode.InternalServerError;

var result = string.Empty;

switch (exception)

{

case ValidationException validationException:

code = HttpStatusCode.BadRequest;

result = JsonSerializer.Serialize(validationException.ValidationResult.ErrorMessage);

break;

case NotFoundException notFoundException:

code = HttpStatusCode.NotFound;

result = JsonSerializer.Serialize(new { error = notFoundException.Message });

\_logger.LogError(

"Error Message: {exceptionMessage}, Time of occurrence {time}",

notFoundException.Message, DateTime.UtcNow);

break;

case AlreadyExistsException alreadyExistsException:

code = HttpStatusCode.BadRequest;

result = JsonSerializer.Serialize(new { error = alreadyExistsException.Message });

\_logger.LogError(

"Error Message: {exceptionMessage}, Time of occurrence {time}",

alreadyExistsException.Message, DateTime.UtcNow);

break;

case LoginException loginException:

code = HttpStatusCode.BadRequest;

result = JsonSerializer.Serialize(new { error = loginException.Message });

\_logger.LogError(

"Error Message: {exceptionMessage}, Time of occurrence {time}",

loginException.Message, DateTime.UtcNow);

break;

default:

code = HttpStatusCode.InternalServerError;

result = JsonSerializer.Serialize(new { error = exception.Message });

\_logger.LogError(

"Error Message: {ex}, Time of occurrence {time}", exception.Message, DateTime.UtcNow);

break;

}

context.Response.ContentType = "application/json";

context.Response.StatusCode = (int)code;

return context.Response.WriteAsync(result);

}

}

1. CustomExceptionHandlerMiddlewareExtentions.cs

namespace TrainTimings.Api.Middlewares;

public static class CustomExceptionHandlerMiddlewareExtentions

{

public static IApplicationBuilder UseCustomExceptionHandler(this IApplicationBuilder builder)

{

return builder.UseMiddleware<CustomExceptionHandlerMiddleware>();

}

}

1. Dockerfile

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base

USER $APP\_UID

WORKDIR /app

EXPOSE 5050

EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build

ARG BUILD\_CONFIGURATION=Release

WORKDIR /src

COPY ["src/TrainTimings.Api/TrainTimings.Api.csproj", "src/TrainTimings.Api/"]

COPY ["src/TrainTimings.Application/TrainTimings.Application.csproj", "src/TrainTimings.Application/"]

COPY ["src/TrainTimings.Core/TrainTimings.Core.csproj", "src/TrainTimings.Core/"]

COPY ["src/TrainTimings.Persistence/TrainTimings.Persistence.csproj", "src/TrainTimings.Persistence/"]

RUN dotnet restore "src/TrainTimings.Api/TrainTimings.Api.csproj"

COPY . .

WORKDIR "/src/src/TrainTimings.Api"

RUN dotnet build "TrainTimings.Api.csproj" -c $BUILD\_CONFIGURATION -o /app/build

FROM build AS publish

ARG BUILD\_CONFIGURATION=Release

RUN dotnet publish "TrainTimings.Api.csproj" -c $BUILD\_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "TrainTimings.Api.dll"]

1. Program.cs

using Serilog;

using TrainTimings.Api.Middlewares;

using TrainTimings.Persistence.Extentions;

var builder = WebApplication.CreateBuilder(args);

var services = builder.Services;

var logger = Log.Logger = new LoggerConfiguration()

.Enrich.FromLogContext()

.WriteTo.Console()

.WriteTo.File($"{Environment.CurrentDirectory}/Logs/{DateTime.UtcNow:yyyy/dd/MM}.txt")

.CreateLogger();

logger.Information("Starting web host");

services.AddControllers();

services.AddEndpointsApiExplorer();

services.AddInfrastructure(builder.Configuration);

services.AddSwaggerGen();

services.AddKeycloakAuthentication(builder.Configuration);

services.AddCustomAuthorization();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseCustomExceptionHandler();

app.UseRouting();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

app.Run();

1. Docker-compose.yml

version: '3.9'

networks:

main\_network:

driver: bridge

services:

traintimings.api:

image: traintimings.api

build:

context: .

dockerfile: src/TrainTimings.Api/Dockerfile

ports:

- 5050:5050

networks:

- main\_network

depends\_on:

- database

- KeycloakTT

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- ASPNETCORE\_URLS=http://+:5050

- ConnectionString=host=postgres-container;port=5432;database=TrainTiming;Username=postgres;Password=toor;

- KeycloakUrl=http://keycloak:8080

KeycloakTT:

image: quay.io/keycloak/keycloak:20.0.2

container\_name: keycloakTT

command:

- start --auto-build --db postgres --hostname-strict-https false --hostname-strict false --proxy edge --http-enabled true --import-realm --spi-user-profile-legacy-user-profile-read-only-attributes \*\_RES\_ACCESS\_MODE

environment:

KC\_DB\_URL: jdbc:postgresql://database:5432/Keycloak

KC\_DB\_USERNAME: postgres

KC\_DB\_PASSWORD: toor

KC\_DB\_SCHEMA: public

KC\_FEATURES: preview

KEYCLOAK\_ADMIN: admin

KEYCLOAK\_ADMIN\_PASSWORD: admin

networks:

- main\_network

ports:

- 9191:8080

depends\_on:

- database

healthcheck:

test: [ "CMD", "curl", "-f", "http://0.0.0.0:8080/realms/master" ]

start\_period: 10s

interval: 30s

retries: 3

timeout: 5s

database:

image: postgres:16.2

container\_name: postgres-container

ports:

- 5555:5432

networks:

main\_network:

healthcheck:

test: pg\_isready -d postgres

interval: 10s

timeout: 5s

retries: 3

start\_period: 5s

environment:

- POSTGRES\_USER=postgres

- POSTGRES\_PASSWORD=toor

- POSTGRES\_DB=Keycloak

volumes:

- postgres-data:/var/lib/postgresql/data

volumes:

postgres-data:

1. AlreadyExistsException.cs

namespace TrainTimings.Application.Exceptions;

/// <summary>

/// Уже существует.

/// </summary>

public class AlreadyExistsException : Exception

{

/// <summary>

/// Конструктор класса.

/// </summary>

/// <param name="name">Название сущности.</param>

/// <param name="key">Уникальное поле сущности.</param>

public AlreadyExistsException(string name, object key)

: base($"Entity {name} with key ({key}) already exists.")

{ }

}

1. BusinessException.cs

namespace TrainTimings.Application.Exceptions;

/// <summary>

/// Бизнес исключение.

/// </summary>

public class BusinessException : Exception

{

/// <summary>

/// Конструктор класса.

/// </summary>

public BusinessException() : base("Что-то пошло не так.")

{ }

}

1. LoginException.cs

namespace TrainTimings.Application.Exceptions;

/// <summary>

/// Ошибка авторизации.

/// </summary>

public class LoginException : Exception

{

/// <summary>

/// Конструктор класса.

/// </summary>

public LoginException() : base("Неверный логин или пароль.")

{ }

}

1. ModelException.cs

using System.ComponentModel.DataAnnotations;

namespace TrainTimings.Application.Exceptions;

/// <summary>

/// Модель некорректна.

/// </summary>

public class ModelException : Exception

{

/// <summary>

/// Список ошибок валидации.

/// </summary>

public List<ValidationResult> Errors { get; set; }

/// <summary>

/// Конструктор.

/// </summary>

/// <param name="validationResults"></param>

public ModelException(List<ValidationResult> validationResults)

{

Errors = validationResults;

}

}

1. NotFoundException.cs

namespace TrainTimings.Application.Exceptions;

/// <summary>

/// Не существует.

/// </summary>

public class NotFoundException : Exception

{

/// <summary>

/// Контруктор класса.

/// </summary>

/// <param name="name">Название сущности.</param>

/// <param name="key">Уникальное поле сущности.</param>

public NotFoundException(string name, object key)

: base($"Entity {name} with key ({key}) not found.")

{ }

}

1. ICitiesTrainRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ICitiesTrainRepository

{

public Task<CitiesTrain> GetCitiesTrainByIdAsync(int id);

public Task<List<CitiesTrain>> GetAllCitiesTrainsAsync();

public Task<CitiesTrain> CreateCitiesTrainAsync(CitiesTrain citiesTrain);

public Task<CitiesTrain> UpdateCitiesTrainAsync(CitiesTrain citiesTrain);

public Task DeleteCitiesTrainAsync(CitiesTrain citiesTrain);

}

1. ICityRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ICityRepository

{

public Task<City> GetCityByIdAsync(int id);

public Task<City> GetCityByNameAsync(string name);

public Task<List<City>> GetAllCitiesAsync();

public Task<City> CreateCityAsync(City city);

public Task<City> UpdateCityAsync(City city);

public Task DeleteCityAsync(City city);

}

1. ITimingsRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ITimingsRepository

{

public Task<Timing> GetTimingByIdAsync(int id);

public Task<List<Timing>> GetAllTimingsAsync();

public Task<Timing> CreateTimingAsync(Timing timing);

public Task<Timing> UpdateTimingAsync(Timing timing);

public Task DeleteTimingAsync(Timing timing);

}

1. ITrainsRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ITrainsRepository

{

public Task<Train> GetTrainByIdAsync(int id);

public Task<Train> GetTrainByNumberAsync(string number);

public Task<List<Train>> GetAllTrainsAsync();

public Task<Train> CreateTrainAsync(Train train);

public Task<Train> UpdateTrainAsync(Train train);

public Task DeleteTrainAsync(Train train);

}

1. ITypesFollowingRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ITypesFollowingRepository

{

public Task<TypesFollowing> GetTypesFollowingByIdAsync(int id);

public Task<List<TypesFollowing>> GetAllTypesFollowingAsync();

public Task<TypesFollowing> CreateTypesFollowingAsync(TypesFollowing typesFollowing);

public Task<TypesFollowing> UpdateTypesFollowingAsync(TypesFollowing typesFollowing);

public Task DeleteTypesFollowingAsync(TypesFollowing typesFollowing);

}

1. ITypesRepository.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IRepository;

public interface ITypesRepository

{

public Task<TypeTrain> GetTypeByIdAsync(int id);

public Task<List<TypeTrain>> GetAllTypesAsync();

public Task<TypeTrain> CreateTypeAsync(TypeTrain typeTrain);

public Task<TypeTrain> UpdateTypeAsync(TypeTrain type);

public Task DeleteTypeAsync(TypeTrain type);

}

1. IAccountService.cs

namespace TrainTimings.Application.Interfaces.IServices;

public interface IAccountService

{

public Task<string> LoginAsync(string username, string password);

public Task ChangePasswordAsync(string username, string oldPassword, string newPassword);

}

1. ICityService.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IServices;

public interface ICityService

{

public Task<List<City>> GetAllAsync();

public Task<City> GetByIdAsync(int id);

public Task<City> CreateAsync(City city);

public Task<City> UpdateAsync(City city);

public Task DeleteAsync(int id);

}

1. ITimingService.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IServices;

public interface ITimingService

{

public Task<Timing> CreateTimingAsync(Timing timing);

public Task<Timing> UpdateTimingAsync(Timing timing);

public Task<Timing> GetTimingByIdAsync(int id);

public Task<List<Timing>> GetAllTimingsAsync();

public Task DeleteTimingAsync(int id);

}

1. ITrainService.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IServices;

public interface ITrainService

{

public Task<Train> GetByIdAsync(int id);

public Task<Train> GetByNumberAsync(string number);

public Task<List<Train>> GetAllAsync();

public Task<Train> CreateAsync(Train train);

public Task<Train> UpdateAsync(Train train);

public Task DeleteAsync(int id);

}

1. ITypeFollowingService.cs

using TrainTimings.Core.Models;

namespace TrainTimings.Application.Interfaces.IServices;

public interface ITypeFollowingService

{

public Task<TypesFollowing> GetAllTypesFollowing();

public Task<TypesFollowing> GetTypesFollowingById(int id);

public Task<TypesFollowing> AddTypesFollowing(TypesFollowing typesFollowing);

public Task<TypesFollowing> UpdateTypesFollowing(TypesFollowing typesFollowing);

public Task<TypesFollowing> DeleteTypesFollowing(int id);

}

1. IUnitOfWork.cs

using TrainTimings.Application.Interfaces.IRepository;

namespace TrainTimings.Application.Interfaces;

public interface IUnitOfWork

{

public ICitiesTrainRepository CitiesTrain { get; }

public ICityRepository City { get; }

public ITimingsRepository Timing { get; }

public ITrainsRepository Train { get; }

public ITypesFollowingRepository TypeFollowing { get; }

public ITypesRepository Type { get; }

}

1. CitiesTrain.cs

namespace TrainTimings.Core.Models;

public partial class CitiesTrain

{

public int Id { get; set; }

public int TrainId { get; set; }

public int CityId { get; set; }

public int TypeId { get; set; }

public virtual City City { get; set; } = null!;

public virtual Train Train { get; set; } = null!;

public virtual TypesFollowing Type { get; set; } = null!;

}

1. City.cs

namespace TrainTimings.Core.Models;

public partial class City

{

public int Id { get; set; }

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

public virtual ICollection<CitiesTrain> CitiesTrains { get; set; } = new List<CitiesTrain>();

}

1. Timing.cs

namespace TrainTimings.Core.Models;

public partial class Timing

{

public int Id { get; set; }

public DateTime Arrival { get; set; }

public DateTime Departure { get; set; }

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

public int TrainId { get; set; }

public virtual Train Train { get; set; } = null!;

}

1. Train.cs

namespace TrainTimings.Core.Models;

public partial class Train

{

public int Id { get; set; }

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

public int TypeId { get; set; }

public virtual ICollection<CitiesTrain> CitiesTrains { get; set; } = new List<CitiesTrain>();

public virtual ICollection<Timing> Timings { get; set; } = new List<Timing>();

public virtual TypeTrain TypeTrain { get; set; } = null!;

}

1. TypesFollowing.cs

namespace TrainTimings.Core.Models;

public partial class TypesFollowing

{

public int Id { get; set; }

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

public DateTime Time { get; set; }

public virtual ICollection<CitiesTrain> CitiesTrains { get; set; } = new List<CitiesTrain>();

}

1. TypeTrain.cs

namespace TrainTimings.Core.Models;

public partial class TypeTrain

{

public int Id { get; set; }

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

public virtual ICollection<Train> Trains { get; set; } = new List<Train>();

}

1. DataContext.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Core.Models;

namespace TrainTimings.Persistence.Data.Context;

public partial class DataContext : DbContext

{

public DataContext()

{

}

public DataContext(DbContextOptions<DataContext> options)

: base(options)

{

}

public virtual DbSet<CitiesTrain> CitiesTrains { get; set; }

public virtual DbSet<City> Cities { get; set; }

public virtual DbSet<Timing> Timings { get; set; }

public virtual DbSet<TypeTrain> TypesTrains { get; set; }

public virtual DbSet<Train> Trains { get; set; }

public virtual DbSet<TypesFollowing> TypesFollowings { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseNpgsql("Server=localhost;port=5555;user id=postgres;password=toor;database=TrainTiming;");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<CitiesTrain>(entity =>

{

entity.HasKey(e => e.Id).HasName("CitiesTrain\_pk");

entity.ToTable("CitiesTrain");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.HasOne(d => d.City).WithMany(p => p.CitiesTrains)

.HasForeignKey(d => d.CityId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("CitiesTrain\_Cities\_Id\_fk");

entity.HasOne(d => d.Train).WithMany(p => p.CitiesTrains)

.HasForeignKey(d => d.TrainId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("CitiesTrain\_Trains\_Id\_fk");

entity.HasOne(d => d.Type).WithMany(p => p.CitiesTrains)

.HasForeignKey(d => d.TypeId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("CitiesTrain\_TypesFollowing\_Id\_fk");

});

modelBuilder.Entity<City>(entity =>

{

entity.HasKey(e => e.Id).HasName("cities\_pk");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.Property(e => e.Name).HasMaxLength(40);

});

modelBuilder.Entity<Timing>(entity =>

{

entity.HasKey(e => e.Id).HasName("Timings\_pk");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.Property(e => e.Arrival).HasColumnType("timestamp without time zone");

entity.Property(e => e.Departure).HasColumnType("timestamp without time zone");

entity.Property(e => e.Platform).HasColumnType("character varying");

entity.HasOne(d => d.Train).WithMany(p => p.Timings)

.HasForeignKey(d => d.TrainId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Timings\_Trains\_Id\_fk");

});

modelBuilder.Entity<TypeTrain>(entity =>

{

entity.HasKey(e => e.Id).HasName("Tipes\_pk");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.Property(e => e.Name).HasMaxLength(30);

});

modelBuilder.Entity<Train>(entity =>

{

entity.HasKey(e => e.Id).HasName("Trains\_pk");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.Property(e => e.Number).HasMaxLength(9);

entity.HasOne(d => d.TypeTrain).WithMany(p => p.Trains)

.HasForeignKey(d => d.TypeId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Trains\_Tipes\_Id\_fk");

});

modelBuilder.Entity<TypesFollowing>(entity =>

{

entity.HasKey(e => e.Id).HasName("TypesFollowing\_pk");

entity.ToTable("TypesFollowing");

entity.Property(e => e.Id).UseIdentityAlwaysColumn();

entity.Property(e => e.Name).HasMaxLength(40);

entity.Property(e => e.Time).HasColumnType("timestamp without time zone");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

1. DiExtentions.cs

using Keycloak.AuthServices.Authentication;

using Keycloak.AuthServices.Authorization;

using Keycloak.AuthServices.Common;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.OpenApi.Models;

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Persistence.Data.Context;

using TrainTimings.Persistence.Helpers;

using TrainTimings.Persistence.Repositories;

using TrainTimings.Persistence.Services;

namespace TrainTimings.Persistence.Extentions;

public static class DiExtentions

{

public static IServiceCollection AddInfrastructure(this IServiceCollection services,

IConfiguration configuration)

{

services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

var connectionString = configuration.GetConnectionString("DefaultConnection");

var dataContext = new DataContext();

dataContext.Database.EnsureCreated();

try

{

dataContext.Database.Migrate();

}

catch (Exception e)

{

Console.WriteLine(e);

}

services.AddDbContext<DataContext>(options => options.UseNpgsql(connectionString));

services.AddScoped<IUnitOfWork, UnitOfWork>();

services.AddScoped<ICitiesTrainRepository, CitiesTrainRepository>();

services.AddScoped<ICityRepository, CityRepository>();

services.AddScoped<ITimingsRepository, TimingsRepository>();

services.AddScoped<ITrainsRepository, TrainsRepository>();

services.AddScoped<ITypesFollowingRepository, TypesFollowingRepository>();

services.AddScoped<ITypesRepository, TypesRepository>();

services.AddScoped<IAccountService, AccountService>();

return services;

}

public static IServiceCollection AddSwaggerGen(this IServiceCollection services)

{

services.AddSwaggerGen(options =>

{

options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

In = ParameterLocation.Header,

Scheme = "Bearer"

});

options.AddSecurityRequirement(new OpenApiSecurityRequirement()

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

},

Scheme = "oauth2",

Name = "Bearer",

In = ParameterLocation.Header

},

new List<string>()

}

});

});

return services;

}

public static IServiceCollection AddKeycloakAuthentication(

this IServiceCollection services, IConfiguration configuration)

{

services.AddKeycloakWebApiAuthentication(configuration, options =>

{

options.RequireHttpsMetadata = false;

});

return services;

}

public static IServiceCollection AddCustomAuthorization(this IServiceCollection services)

{

services

.AddAuthorization()

.AddKeycloakAuthorization(options =>

{

options.EnableRolesMapping = RolesClaimTransformationSource.Realm;

options.RoleClaimType = KeycloakConstants.RoleClaimType;

})

.AddAuthorizationBuilder()

.AddPolicy(

"AdminPolicy",

policy => policy.RequireRole("Admin"))

.AddPolicy(

"DispatcherPolicy",

policy => policy.RequireRole("Dispatcher"))

.AddPolicy("MachinistPolicy",

policy => policy.RequireRole("Machinist"));

return services;

}

}

1. HttpClientHelper.cs

namespace TrainTimings.Persistence.Helpers;

public class HttpClientHelper

{

private static HttpClient \_httpClient;

public static HttpClient GetHttpClient() =>

\_httpClient ??= new HttpClient();

}

1. UnitOfWork.cs

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Helpers;

public class UnitOfWork : IUnitOfWork

{

private readonly DataContext \_dbContext;

private readonly ICityRepository \_cityRepository;

private readonly ICitiesTrainRepository \_citiesTrainRepository;

private readonly ITimingsRepository \_timingsRepository;

private readonly ITrainsRepository \_trainsRepository;

private readonly ITypesRepository \_typesRepository;

private readonly ITypesFollowingRepository \_typesFollowingRepository;

public UnitOfWork(DataContext dbContext, ICityRepository cityRepository,

ICitiesTrainRepository citiesTrainRepository, ITimingsRepository timingsRepository,

ITrainsRepository trainsRepository, ITypesRepository typesRepository,

ITypesFollowingRepository typesFollowingRepository)

{

\_dbContext = dbContext;

\_cityRepository = cityRepository;

\_citiesTrainRepository = citiesTrainRepository;

\_timingsRepository = timingsRepository;

\_trainsRepository = trainsRepository;

\_typesRepository = typesRepository;

\_typesFollowingRepository = typesFollowingRepository;

}

public ICitiesTrainRepository CitiesTrain { get => \_citiesTrainRepository; }

public ICityRepository City { get => \_cityRepository; }

public ITimingsRepository Timing { get => \_timingsRepository; }

public ITrainsRepository Train { get => \_trainsRepository; }

public ITypesFollowingRepository TypeFollowing { get => \_typesFollowingRepository; }

public ITypesRepository Type { get => \_typesRepository; }

}

1. CitiesTrainRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class CitiesTrainRepository : ICitiesTrainRepository

{

private readonly DataContext \_dataContext;

public CitiesTrainRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<CitiesTrain> GetCitiesTrainByIdAsync(int id)

{

return await \_dataContext.CitiesTrains.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<List<CitiesTrain>> GetAllCitiesTrainsAsync()

{

return await \_dataContext.CitiesTrains.ToListAsync();

}

public async Task<CitiesTrain> CreateCitiesTrainAsync(CitiesTrain citiesTrain)

{

await \_dataContext.CitiesTrains.AddAsync(citiesTrain);

await \_dataContext.SaveChangesAsync();

return citiesTrain;

}

public async Task<CitiesTrain> UpdateCitiesTrainAsync(CitiesTrain citiesTrain)

{

\_dataContext.CitiesTrains.Update(citiesTrain);

await \_dataContext.SaveChangesAsync();

return citiesTrain;

}

public async Task DeleteCitiesTrainAsync(CitiesTrain citiesTrain)

{

\_dataContext.CitiesTrains.Remove(citiesTrain);

await \_dataContext.SaveChangesAsync();

}

}

1. CityRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class CityRepository : ICityRepository

{

private readonly DataContext \_dataContext;

public CityRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<City> GetCityByIdAsync(int id)

{

return await \_dataContext.Cities.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<City> GetCityByNameAsync(string name)

{

return await \_dataContext.Cities.FirstOrDefaultAsync(x => x.Name == name);

}

public async Task<List<City>> GetAllCitiesAsync()

{

return await \_dataContext.Cities.ToListAsync();

}

public async Task<City> CreateCityAsync(City city)

{

await \_dataContext.Cities.AddAsync(city);

await \_dataContext.SaveChangesAsync();

return city;

}

public async Task<City> UpdateCityAsync(City city)

{

\_dataContext.Cities.Update(city);

await \_dataContext.SaveChangesAsync();

return city;

}

public async Task DeleteCityAsync(City city)

{

\_dataContext.Cities.Remove(city);

await \_dataContext.SaveChangesAsync();

}

}

1. TimingsRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class TimingsRepository : ITimingsRepository

{

private readonly DataContext \_dataContext;

public TimingsRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<Timing> GetTimingByIdAsync(int id)

{

return await \_dataContext.Timings.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<List<Timing>> GetAllTimingsAsync()

{

return await \_dataContext.Timings.ToListAsync();

}

public async Task<Timing> CreateTimingAsync(Timing timing)

{

await \_dataContext.Timings.AddAsync(timing);

await \_dataContext.SaveChangesAsync();

return timing;

}

public async Task<Timing> UpdateTimingAsync(Timing timing)

{

\_dataContext.Timings.Update(timing);

await \_dataContext.SaveChangesAsync();

return timing;

}

public async Task DeleteTimingAsync(Timing timing)

{

\_dataContext.Timings.Remove(timing);

await \_dataContext.SaveChangesAsync();

}

}

1. TrainsRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class TrainsRepository : ITrainsRepository

{

private readonly DataContext \_dataContext;

public TrainsRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<Train> GetTrainByIdAsync(int id)

{

return await \_dataContext.Trains.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<Train> GetTrainByNumberAsync(string number)

{

return await \_dataContext.Trains.FirstOrDefaultAsync(x => x.Number == number);

}

public async Task<List<Train>> GetAllTrainsAsync()

{

return await \_dataContext.Trains.ToListAsync();

}

public async Task<Train> CreateTrainAsync(Train train)

{

await \_dataContext.Trains.AddAsync(train);

await \_dataContext.SaveChangesAsync();

return train;

}

public async Task<Train> UpdateTrainAsync(Train train)

{

\_dataContext.Trains.Update(train);

await \_dataContext.SaveChangesAsync();

return train;

}

public async Task DeleteTrainAsync(Train train)

{

\_dataContext.Trains.Remove(train);

await \_dataContext.SaveChangesAsync();

}

}

1. TypesFollowingRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class TypesFollowingRepository : ITypesFollowingRepository

{

private readonly DataContext \_dataContext;

public TypesFollowingRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<TypesFollowing> GetTypesFollowingByIdAsync(int id)

{

return await \_dataContext.TypesFollowings.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<List<TypesFollowing>> GetAllTypesFollowingAsync()

{

return await \_dataContext.TypesFollowings.ToListAsync();

}

public async Task<TypesFollowing> CreateTypesFollowingAsync(TypesFollowing typesFollowing)

{

await \_dataContext.TypesFollowings.AddAsync(typesFollowing);

await \_dataContext.SaveChangesAsync();

return typesFollowing;

}

public async Task<TypesFollowing> UpdateTypesFollowingAsync(TypesFollowing typesFollowing)

{

\_dataContext.TypesFollowings.Update(typesFollowing);

await \_dataContext.SaveChangesAsync();

return typesFollowing;

}

public async Task DeleteTypesFollowingAsync(TypesFollowing typesFollowing)

{

\_dataContext.TypesFollowings.Remove(typesFollowing);

await \_dataContext.SaveChangesAsync();

}

}

1. TypesRepository.cs

using Microsoft.EntityFrameworkCore;

using TrainTimings.Application.Interfaces.IRepository;

using TrainTimings.Core.Models;

using TrainTimings.Persistence.Data.Context;

namespace TrainTimings.Persistence.Repositories;

public class TypesRepository : ITypesRepository

{

private readonly DataContext \_dataContext;

public TypesRepository(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task<TypeTrain> GetTypeByIdAsync(int id)

{

return await \_dataContext.TypesTrains.FirstOrDefaultAsync(x => x.Id == id);

}

public async Task<List<TypeTrain>> GetAllTypesAsync()

{

return await \_dataContext.TypesTrains.ToListAsync();

}

public async Task<TypeTrain> CreateTypeAsync(TypeTrain typeTrain)

{

await \_dataContext.TypesTrains.AddAsync(typeTrain);

await \_dataContext.SaveChangesAsync();

return typeTrain;

}

public async Task<TypeTrain> UpdateTypeAsync(TypeTrain type)

{

\_dataContext.TypesTrains.Update(type);

await \_dataContext.SaveChangesAsync();

return type;

}

public async Task DeleteTypeAsync(TypeTrain type)

{

\_dataContext.Remove(type);

await \_dataContext.SaveChangesAsync();

}

}

1. AccountService.cs

using System.Net.Http.Headers;

using System.Text;

using Microsoft.Extensions.Configuration;

using Newtonsoft.Json.Linq;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Persistence.Helpers;

namespace TrainTimings.Persistence.Services;

public class AccountService : IAccountService

{

private static HttpClient client = HttpClientHelper.GetHttpClient();

private readonly IConfiguration \_configuration;

public AccountService(IConfiguration configuration)

{

\_configuration = configuration;

}

public async Task<string> LoginAsync(string username, string password)

{

var reqestKeycloak = new Dictionary<string, string>

{

{"grant\_type", \_configuration["KeycloakLoginRequest:grant\_type"]},

{"client\_id", \_configuration["KeycloakLoginRequest:client\_id"]},

{"username", username},

{"password", password},

{"client\_secret", \_configuration["KeycloakLoginRequest:client\_secret"]},

{"scope", \_configuration["KeycloakLoginRequest:scope"]}

};

var response = await client.PostAsync(\_configuration["KeycloakLoginRequest:url"],

new FormUrlEncodedContent(reqestKeycloak));

var responseString = JObject.Parse(await response.Content.ReadAsStringAsync());

var token = (string)responseString["access\_token"];

return token;

}

public async Task ChangePasswordAsync(string username, string oldPassword, string newPassword)

{

var token = await LoginAsync(username, oldPassword);

var url = $"http://keycloak-server/auth/admin/realms/MyRealm/users/{username}";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

var content = new StringContent($"password={newPassword}", Encoding.UTF8, "application/x-www-form-urlencoded");

var response = await client.PutAsync(url, content);

}

}

1. CityService.cs

using TrainTimings.Application.Exceptions;

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Persistence.Services;

public class CityService : ICityService

{

private readonly IUnitOfWork \_unitOfWork;

public CityService(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public async Task<List<City>> GetAllAsync()

{

return await \_unitOfWork.City.GetAllCitiesAsync();

}

public async Task<City> GetByIdAsync(int id)

{

return await \_unitOfWork.City.GetCityByIdAsync(id);

}

public async Task<City> CreateAsync(City city)

{

var identity = await \_unitOfWork.City.GetCityByNameAsync(city.Name);

if (identity != null)

throw new AlreadyExistsException(nameof(City), city.Name);

var creatingCity = new City

{

Name = city.Name

};

var createdCity = await \_unitOfWork.City.CreateCityAsync(creatingCity);

return createdCity;

}

public async Task<City> UpdateAsync(City city)

{

var updatingCity = await \_unitOfWork.City.GetCityByIdAsync(city.Id);

if (updatingCity == null)

throw new NotFoundException(nameof(City), city.Id);

updatingCity.Name = city.Name;

var updatedCity = await \_unitOfWork.City.UpdateCityAsync(updatingCity);

return updatedCity;

}

public async Task DeleteAsync(int id)

{

var deletingCity = await \_unitOfWork.City.GetCityByIdAsync(id);

if (deletingCity == null)

throw new NotFoundException(nameof(City), id);

await \_unitOfWork.City.DeleteCityAsync(deletingCity);

}

}

1. TimingService.cs

using TrainTimings.Application.Exceptions;

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Persistence.Services;

public class TimingService : ITimingService

{

private readonly IUnitOfWork \_unitOfWork;

public TimingService(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public async Task<Timing> CreateTimingAsync(Timing timing)

{

var train = await \_unitOfWork.Train.GetTrainByIdAsync(timing.TrainId);

if (train == null)

throw new NotFoundException(nameof(Train), timing.TrainId);

var createdTiming = await \_unitOfWork.Timing.CreateTimingAsync(timing);

return createdTiming;

}

public async Task<Timing> UpdateTimingAsync(Timing timing)

{

var train = await \_unitOfWork.Train.GetTrainByIdAsync(timing.TrainId);

if (train == null)

throw new NotFoundException(nameof(Train), timing.TrainId);

var timingToUpdate = await \_unitOfWork.Timing.GetTimingByIdAsync(timing.Id);

if (timingToUpdate == null)

throw new NotFoundException(nameof(Timing), timing.Id);

timingToUpdate.Arrival = timing.Arrival;

timingToUpdate.Departure = timing.Departure;

var updatedTiming = await \_unitOfWork.Timing.UpdateTimingAsync(timingToUpdate);

return updatedTiming;

}

public async Task<Timing> GetTimingByIdAsync(int id)

{

var timing = await \_unitOfWork.Timing.GetTimingByIdAsync(id);

if (timing == null)

throw new NotFoundException(nameof(Timing), id);

return timing;

}

public async Task<List<Timing>> GetAllTimingsAsync()

{

return await \_unitOfWork.Timing.GetAllTimingsAsync();

}

public async Task DeleteTimingAsync(int id)

{

var timing = await \_unitOfWork.Timing.GetTimingByIdAsync(id);

if (timing == null)

throw new NotFoundException(nameof(Timing), id);

await \_unitOfWork.Timing.DeleteTimingAsync(timing);

}

}

1. TrainService.cs

using TrainTimings.Application.Exceptions;

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Persistence.Services;

public class TrainService : ITrainService

{

private readonly IUnitOfWork \_unitOfWork;

public TrainService(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public async Task<Train> GetByIdAsync(int id)

{

return await \_unitOfWork.Train.GetTrainByIdAsync(id);

}

public async Task<Train> GetByNumberAsync(string number)

{

return await \_unitOfWork.Train.GetTrainByNumberAsync(number);

}

public async Task<List<Train>> GetAllAsync()

{

return await \_unitOfWork.Train.GetAllTrainsAsync();

}

public async Task<Train> CreateAsync(Train train)

{

var identity = await \_unitOfWork.Train.GetTrainByNumberAsync(train.Number);

if (identity != null)

throw new AlreadyExistsException(nameof(Train), train.Number);

var newTrain = new Train

{

Number = train.Number

};

var createdTrain = await \_unitOfWork.Train.CreateTrainAsync(newTrain);

return createdTrain;

}

public async Task<Train> UpdateAsync(Train train)

{

var updatingTrain = await \_unitOfWork.Train.GetTrainByIdAsync(train.Id);

if (updatingTrain == null)

throw new NotFoundException(nameof(Train), train.Id);

updatingTrain.Number = train.Number;

var updatedTrain = await \_unitOfWork.Train.UpdateTrainAsync(updatingTrain);

return updatedTrain;

}

public async Task DeleteAsync(int id)

{

var deletingTrain = await \_unitOfWork.Train.GetTrainByIdAsync(id);

if (deletingTrain == null)

throw new NotFoundException(nameof(Train), id);

await \_unitOfWork.Train.DeleteTrainAsync(deletingTrain);

}

}

1. TypeFollowingService.cs

using TrainTimings.Application.Interfaces;

using TrainTimings.Application.Interfaces.IServices;

using TrainTimings.Core.Models;

namespace TrainTimings.Persistence.Services;

public class TypeFollowingService : ITypeFollowingService

{

private readonly IUnitOfWork \_unitOfWork;

public TypeFollowingService(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public async Task<TypesFollowing> GetAllTypesFollowing()

{

throw new NotImplementedException();

}

public async Task<TypesFollowing> GetTypesFollowingById(int id)

{

throw new NotImplementedException();

}

public async Task<TypesFollowing> AddTypesFollowing(TypesFollowing typesFollowing)

{

throw new NotImplementedException();

}

public async Task<TypesFollowing> UpdateTypesFollowing(TypesFollowing typesFollowing)

{

throw new NotImplementedException();

}

public async Task<TypesFollowing> DeleteTypesFollowing(int id)

{

throw new NotImplementedException();

}

}

# Результаты работы программы