Содержание

[1. ВВЕДЕНИЕ 4](#_Toc162988970)

[1.1. Наименование программы 4](#_Toc162988971)

[1.2. Документы, на основании которых ведется разработка 4](#_Toc162988972)

[2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 5](#_Toc162988973)

[2.1. Функциональное назначение 5](#_Toc162988974)

[2.2. Эксплуатационное назначение 5](#_Toc162988975)

[2.3. Краткая характеристика области применения программы 5](#_Toc162988976)

[3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 6](#_Toc162988977)

[3.1. Постановка задачи на разработку программы 6](#_Toc162988978)

[3.2. Описание функционирования программы 8](#_Toc162988979)

[3.2.1. Изначальная архитектура базы данных 9](#_Toc162988980)

[3.2.2. Описание архитектуры приложения 9](#_Toc162988981)

[3.2.3. Описание реализации авторизации 10](#_Toc162988982)

[3.2.4. Описание реализации взаимодействия с базой данных 11](#_Toc162988983)

[3.2.5. Описание маппинга моделей 12](#_Toc162988984)

[3.2.6. Описание реализации бизнес-сервисов 12](#_Toc162988985)

[3.2.7. Описание реализации валидации данных 12](#_Toc162988986)

[3.2.8. Описание реализации инфраструктуры 13](#_Toc162988987)

[3.2.9. Описание реализации конфигурации 13](#_Toc162988988)

[3.2.10. Описание реализации Api 14](#_Toc162988989)

[3.2.11. Описание реализации интеграции Swager 14](#_Toc162988990)

[3.2.12. Описание работы с Docker 15](#_Toc162988991)

[3.3. Описание и обоснование выбора состава технических и программных средств 18](#_Toc162988992)

[3.3.1. Состав технических и программных средств 18](#_Toc162988993)

[3.3.2. Обоснование выбора состава технических и программных средств 18](#_Toc162988994)

[3.4. Описание и обоснование выбора метода организации входных и выходных данных 19](#_Toc162988995)

[Входные данные получаются из Http запросов к Api. Либо в виде json тела запроса, либо из параметров запроса. 19](#_Toc162988996)

[4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 20](#_Toc162988997)

[4.1. Ориентировочная экономическая эффективность 20](#_Toc162988998)

[4.2. Предполагаемая потребность 20](#_Toc162988999)

[4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами 20](#_Toc162989000)

[ПРИЛОЖЕНИЕ 1 27](#_Toc162989001)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 31](#_Toc162989002)

# ВВЕДЕНИЕ

# Наименование программы

**Наименование программы** – «КофеСкаут».

**Условное обозначение темы разработки** – «CoffeeScout».

# Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем тема курсового проекта.

**Организация, утвердившая этот документ:** НИУ “Высшая школа экономики”, Факультет компьютерных наук, ОП “Программная инженерия”.

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

# Функциональное назначение

Backend приложения реализованный в виде REST API, предназначен для работы с данными из базы данных и обработки бизнес процессов: заказ напитков, поиск кофеен и напитков.

# Эксплуатационное назначение

API предоставляет возможности для авторизации, размещения заказов пользователями,

получения заказов и редактирования меню кофейнями, поиска кофеен и напитков с различными фильтрами.

# Краткая характеристика области применения программы

«КофеСкаут» - приложение для поиска интересующего напитка в кофейнях,

расположенных в непосредственной близости от пользователя.

Основная область применения программы - личное использование. Также приложение

предоставляет возможность кофейням продвигать свои продукты и рекламировать себя.

Клиентам приложение предоставляет возможность выбора желаемого напитка из

предложенных с использованием системы фильтров, а после предлагает ближайшие к

пользователю кофейни, которые смогут приготовить выбранный напиток, а также

предлагает сделать предзаказ. Более того, приложение указывает кратчайший путь до

выбранной кофейни, что позволит пользователям сэкономить время как на выборе кофейни,

так и на времени, чтобы до неё добраться.

В нынешних реалиях с большим количеством кофеен и ассортимента в них бывает

трудно найти желаемый напиток поблизости, поэтому приложение «КофеСкаут» будет

пользоваться спросом.

# ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

# Постановка задачи на разработку программы

В результате разработки backend’а приложения должен быть получен REST API со следующими endpoint’ами:

1. **Accounts:**

* **[POST] /accounts/customer/register:** регистрация пользователя мобильного приложения.
* **[POST] /accounts/cafe-admin/register:** регистрация администратора кофейни. Доступно только администратору серверной части приложения.
* **[POST] /accounts/login:** авторизация зарегистрированного пользователя.
* **[POST] /accounts/refresh:** обновление истекшего токена авторизации.
* **[GET] /accounts/confirmEmail:** подтверждение адреса электронной почты.
* **[POST] /accounts/resendConfirmationEmail:** позволяет повторно прислать письмо с для подтверждения адреса электронной почты.
* **[POST] /accounts/forgotPassword:** присылает письмо, содержащее код для смены пароля.
* **[POST] /accounts/resetPassword:** позволяет сменить пароль в случае, если пользователь не помнит старый.
* **[GET] /accounts/manage/info:** позволяет получить информацию о пользователе: адрес электронной почты, статус подтверждения почты.
* **[POST] /accounts/manage/info:** позволяет изменить данные о пользователе: Почту, пароль.

1. **BeverageTypes:**

* **[POST] /beverage-types:** добавить тип напитка в базу данных. Доступно только администратору серверной части приложения.
* **[PATCH] /beverage-types:** обновить тип напитка в базе данных. Доступно только администратору серверной части приложения.
* **[DELETE] /beverage-types:** удалить тип напитка из базы данных. Доступно только администратору серверной части приложения.

1. **Cafes:**

* **[POST] /cafes:** добавить кофейню в базу данных. Доступно только администратору серверной части приложения.
* **[DELETE] /cafes/{id}:** удалить кофейню из базы данных. Доступно только администратору серверной части приложения.
* **[PATCH] /cafes/{id}:** обновить данные о кофейне. Доступно только администратору кофейни.
* **[GET] /cafes:** получить кофейни из базы данных в соответствии с переданными фильтрами. Доступно только пользователю мобильного приложения.
* **[GET] /cafes/orders:** получить заказы, связанные с кофейней в соответствии с фильтрами. Доступно только администратору кофейни.

1. **Customers:**

* **[POST] /customers/favored-menu-items:** добавить напиток в любимое. Доступно только пользователю мобильного приложения.
* **[GET] /customers/favored-menu-items:** получить любимые напитки из базы данных. Доступно только пользователю мобильного приложения.
* **[GET] /customers/favored-beverage-types:** получить типы любимых напитков из базы данных. Доступно только пользователю мобильного приложения.
* **[GET] /customers/orders:** получить из базы данных заказы, связанные с пользователем. Доступно только пользователю мобильного приложения.

1. **MenuItems:**

* **[GET] /menu-items:** получить напитки из базы данных в соответствии с фильтрами. Доступно только пользователю мобильного приложения.
* **[POST] /menu-items:** добавить напиток в базу данных с учетом принадлежности к кофейне. Доступно только администратору кофейни.
* **[PATCH] /menu-items/{id}:** изменить информацию о напитке в базе данных. Доступно только администратору кофейни.
* **[DELETE] /menu-items/{id}:** удалить напиток из базы данных. Доступно только администратору кофейни.
* **[POST] /menu-items/{id}/reviews:** добавить отзыв к напитку в базу данных. Доступно только пользователю мобильного приложения, который заказывал данный напиток.
* **[PATCH] /menu-items/{id}/reviews/{review\_id}:** изменить содержание отзыва. Доступно только пользователю мобильного приложения, разместившему отзыв.
* **[DELETE] /menu-items/{id}/reviews/{review\_id}:** удалить отзыв к напитку из базы данных. Доступно только пользователю мобильного приложения, разместившему отзыв.

1. **Orders:**

* **[POST] /orders:** разместить заказ. Доступно только пользователю мобильного приложения.
* **[PATCH] /orders/{id}/complete:** оповестить о готовности части заказа, связанной с кофейней. Доступно только администратору кофейни.
* **[PATCH] /orders/{id}/cancel:** изменить статус заказа на отмененный. Доступно пользователю мобильного приложения и администратору кофейни.
* **[PATCH] /orders/{id}/pay:** оплатить заказ и изменить статус заказа на оплаченный. Доступно пользователю мобильного приложения.
* **[GET] /orders/{id}:** получить заказ из базы данных. Доступно пользователю мобильного приложения, если заказ с **id** принадлежит ему. Доступно администратору кофейни, если заказ с **id** связан с кофейней. Доступно администратору серверной части приложения.

# Описание функционирования программы

Backend приложения написан на языке С# с использованием фреймворка ASP.NET. Архитектура приложения разработана в соответствии с принципами Domain Driven Design и Clean architecture. В качестве базы данных выбрана [PostgresSQL](#postgres) с расширением [Postgis](#postgis) для работы с географическими данными. Для развертывания приложения используется система контейнеризации Docker.

# Изначальная архитектура базы данных

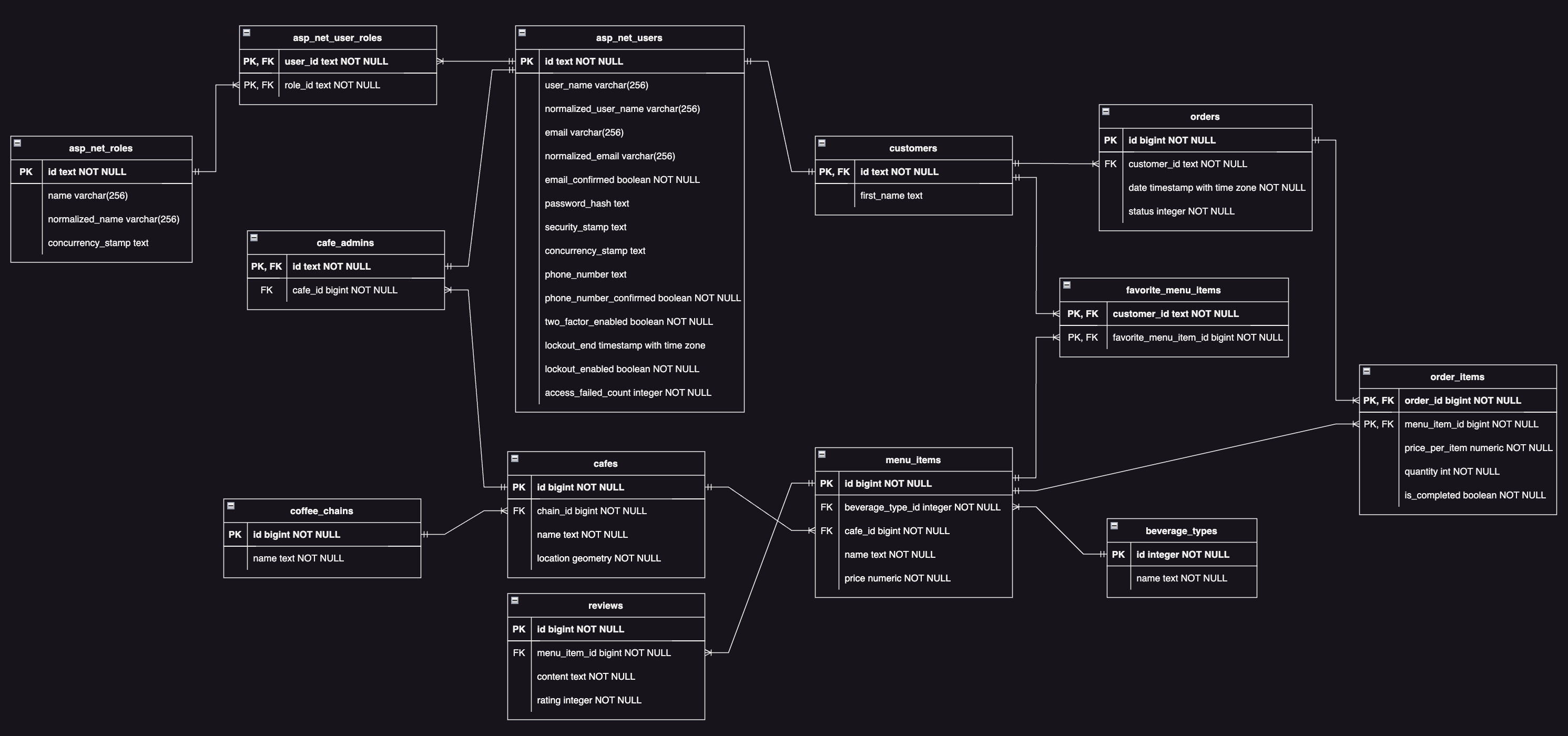


Рисунок 1 – Entity Relationship Diagram для базы данных.

[Ссылка на картинку в большем размере.](https://ibb.co/2Y2sGR4)

# Описание архитектуры приложения

Приложение состоит из четырех проектов, между которыми существуют следующие зависимости. Приложение состоит из четырех проектов, между которыми существуют следующие зависимости.

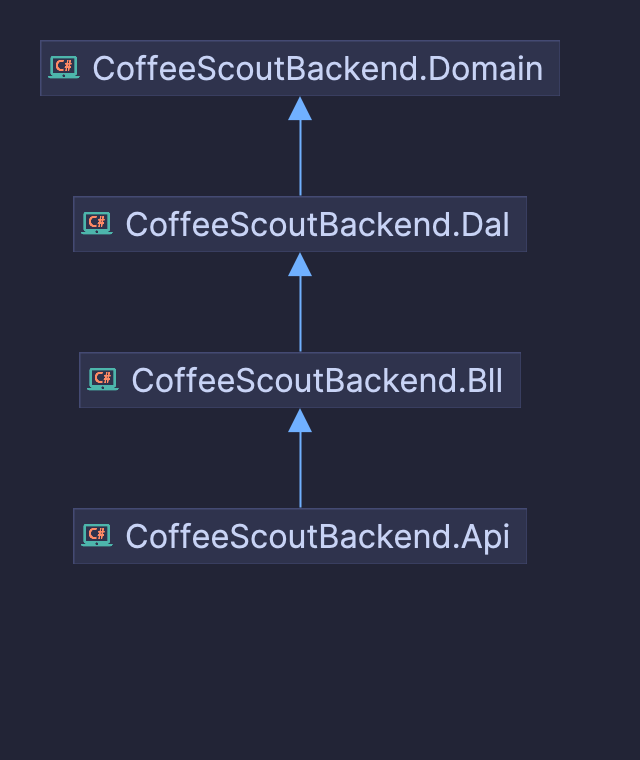


Рисунок 2 – Схема зависимостей.

* **Domain:** Ядро приложения. Содержит в себе бизнес-модели, интерфейсы сервисов и репозиториев, а также исключения, связанные с бизнес-правилами.
* **Bll:** Слой, содержащий реализации бизнес сервисов и нужную им инфраструктуру.
* **Dal:** Слой, содержащий реализации репозиториев и нужную им инфраструктуру. Отвечает за работу с базой данных.
* **Api:** Слой, содержащий контроллеры, инфраструктуру, валидацию запросов, модели ответов и запросов. Слой отвечает за взаимодействие серверной части с внешним миром через http. В нем происходит конфигурация и запуск приложения.

Для избежания сильной связности между слоями бизнес-модели имеют свои аналоги в слое

Dal и Api. Это разделение позволило исключить из Domain слоя любые внешние зависимости в соответствии с принципами Clean architecture. Все слои приложения взаимодействуют друг с другом только посредством интерфейсов, что вместе с использованием механизма Dependency Injection позволило добиться прозрачной и понятной архитектуры.

# Описание реализации авторизации

Для реализации регистрации и авторизации используется библиотека **ASP.NET Core Identity**. Она упрощает процесс интеграции авторизации в приложение путем предоставления различных готовых решений и сервисов. Это позволяет сильно ускорить разработку. В проекте используется стандартная конфигурация Identity с некоторыми нужными модификациями. Написано свое решение для регистрации пользователей с учетом роли – RoleRegistrationService. Выполнена конфигурация Identity сервисов под бизнес требования: наложены ограничения на допустимые логины и пароли, количество максимальных неудачных попыток авторизации, время жизни токена.

# Описание реализации взаимодействия с базой данных

Для более удобной дальнейшей интеграции Identity было принято решение использовать ORM[**EntityFrameworkCore**](#entityFrameworkCore)**.**  Данное решение позволяет писать запросы к базе данных, не используя SQL напрямую, что сильно упрощает реализацию взаимодействия с базой данных. В масштабах приложения снижение производительности запросов нивелируется ускорением разработки репозиториев. Использование **EntityFrameworkCore** также позволило упростить процесс миграций, за счет их автоматической генерации на основе сущностей. Генерация миграций происходит с использованием библиотеки [**Microsoft.EntityFrameworkCore.Design**](#entityFrameworkCore). В процессе анализа бизнес-требований стала понятна необходимость использования гео-типов в базе данных. Для работы с ними внутри C# кода была использована библиотека [**Npgsql.EntityFrameworkCore.PostgreSQL.NetTopologySuite**](#npgsql)**,** интегрированная с **EntityFrameworkCore** и позволяющая использовать различные географические функции. Это было нужно для написания запросов в базу данных, которые возвращали бы сущности в радиусе от определенной точки. Подключение к базе данных производится посредством использования библиотеки [**Npgsql.EntityFrameworkCore.PostgreSQL**](#npgsql) предоставляющей возможность работать с Postgres через **EntityFrameworkCore.** Так как **EntityFrameworkCore** изначально использует названия свойств сущностей для создания столбцов и таблиц, то была нарушена политика наименования, принятая в Postgres. Чтобы изменить стандартную политику наименования **EntityFrameworkCore,** была использована библиотека [**EFCore.NamingConventions**](#EFNamingConventions). Это не помогло изменить названия самих таблиц, поэтому также было написано свое решение, включенное в конфигурацию **EntityFrameworkCore.** Классы, отвечающие за конфигурацию генерируемых таблиц: ModelBuilderExtensions и AppDbContext. Подключение к базе данных происходит с помощью строки подключения, полученной из конфигурации. В строке конфигурации указано Pooling=true;, что позволяет использовать повторно созданные подключения к базе данных. Классом-провайдером конфигурации является DatabaseSettings. Для непосредственной инкапсуляции логики получения данных были написаны классы репозиториев. Для отправки запросов к базе они используют класс AppDbContext.

# Описание маппинга моделей

Так как в проекте для каждого слоя использовались свои сущности, встал вопрос об использовании готового решения, позволяющего автоматически маппить модели одного слоя в модели другого. Была выбрана библиотека [**Mapster**](#mapster)**,** за ее легковесность и простоту конфигурации. Из-за того, что в некоторых моделях возникали рекурсивные зависимости, пришлось также производить дополнительную конфигурацию некоторых преобразований. Это позволило избежать ошибки Stack Overflow при маппинге некоторых моделей. Использование библиотеки уменьшило количество boilerplate кода внутри контроллеров, сервисов и репозиториев, что сделало код более наглядным.

# Описание реализации бизнес-сервисов

Для того, чтобы сделать архитектуру более прозрачной было принято решение, под каждый репозиторий писать сервис. Это позволило оставить в репозиториях, только логику получения данных, а применение любых бизнес-правил к полученным данным вынести в сервисы. Благодаря этому, получилось упростить код и в сервисах, которые используют другие сервисы для своей работы.

# Описание реализации валидации данных

Для того чтобы производить валидацию входящих запросов было принято решение использовать библиотеку [**FluentValidation.AspNetCore**](#fluentValidation).Она позволила написать классы валидации, которые бы применяли ограничения к входным данным в соответствии с бизнес-требованиями. **FluentValidation.AspNetCore** не предоставляет возможности реализации автоматической асинхронной валидации. Использование синхронного варианта могло стать узким горлышком приложения, поэтому возникла потребность в реализации своего решения. В ASP.NET при использовании Http контролеров достаточно простого решения написать возможности не было из-за особенностей обработки Http запросов в ASP.NET. В документации к **FluentValidation.AspNetCore** предлагалось использовать библиотеку [**SharpGrip.FluentValidation.AutoValidation.Mvc**](#sharpGrip) для реализации асинхронной автоматической валидации. Было принято решение использовать ее вместо написания своего более сложного и менее эффективного решения.

# Описание реализации инфраструктуры

В качестве инфраструктурных решений были реализованы классы для упрощения работы с типами NetTopologySuite и временем: GpsLocationProvider – создает Point с заданным подходящим для gps SRID и Geometry, представляющий из себя область заданого радиуса, DatetimeProvider – инкапсулирует получение DateTime объектов. Также был реализован интерфейс IDbSeeder, который позволяет предоставить реализацию для заполнения базы данных изначальными данными, такими как данные администратора серверной части.

# Описание реализации конфигурации

В конфигурацию приложения входит несколько аспектов. Через конфигурацию получается строка подключения к базе данных и данные для авторизации администратора серверной части приложения. Для безопасного хранения этих данных во время разработки использовался механизм .NET Secrets, который позволяет отделить хранение секретных конфигурационных файлов от проекта. Чтобы использовать данные конфигурации внутри C# кода был использован стандартный паттерн Asp.Net по маппингу данных в классы-провайдеры конфигурации. .NET Secrets помогают скрыть данные во время разработки, но не пригодны для использования в Production среде. Изучив различные способы передачи секретов в Production в контексте проекта, было принято решения использовать переменные окружения. Стандартный механизм конфигурации Asp.Net позволяет маппить значения из переменных окружения в те же классы-провайдеры. Для упрощения передачи переменных окружения был создан .env файл используемый при сборке проекта в контейнер Docker. Пример файла конфигурации secrets.json и .env:

****

Рисунок 3 – secrets.json.

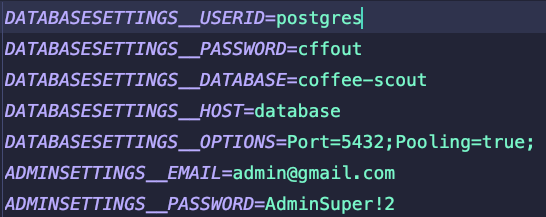


Рисунок 4 - .env.

# Описание реализации Api

Каждый метод контролера возвращает IActionResult. Этот тип позволяет возвращать обертки над результатами с Http кодами. В случае успешного выполнения метода, возвращается один из кодов 2\*\*. Если во время выполнения запроса возникает ошибка, то возвращается либо код 4\*\* и тело ответа содержит описание ошибки в соответствии с соглашением Problem details, либо код 5\*\*, обозначающий внутреннюю ошибку сервера. Для логирования во время разработки общих деталей запроса и ответа использован стандартный логирующий middleware.

# Описание реализации интеграции Swager

Для описания Api используется спецификация [OpenApi](#openApi) и Swager. Для поддержки подробного описания ответа в Swager каждый метод контролера помечен атрибутом [ProducesResponseType] с указанием возвращаемого типа и Http кода. Пример:



Рисунок 5 – пример метода контролера.

Чтобы обеспечить поддержку авторизации в Swager, была написана дополнительная конфигурация:



Рисунок 6 – конфигурация Swager.

# Описание работы с Docker

Было принято решение для разворачивания Backend’а приложения использовать систему контейнеризации [Docker](#docker). Для работы серверной части нужно было настроить взаимодействие двух контейнеров: контейнер с приложением, контейнер для базы данных. Так как требовалось запустить 2 связанных контейнера я выбрал инструмент [Docker compose](#dockerCompose) для управления мульти контейнерными окружениями. Был разработан следующий docker-compose.yml:

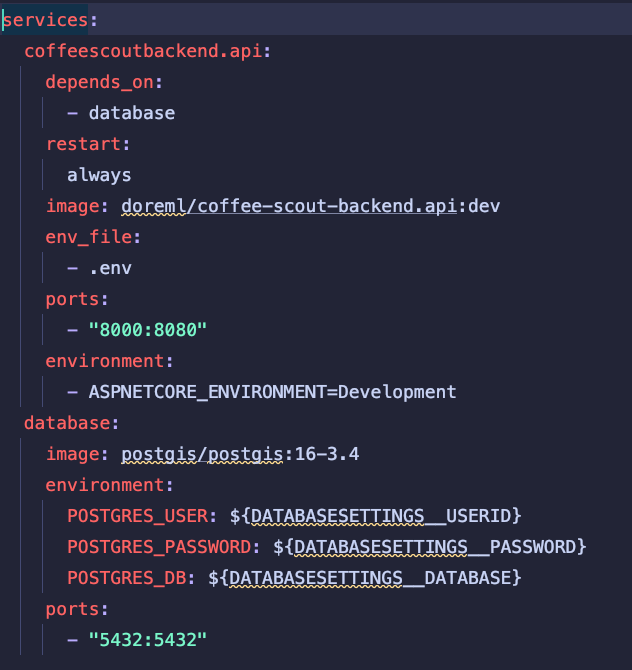


Рисунок 7 – docker-compose.yml.

Для создания контейнера с приложением используется образ, собранный с использованием Dockerfile и загруженный в личный репозиторий на [Docker registry](#dockerRegistry).

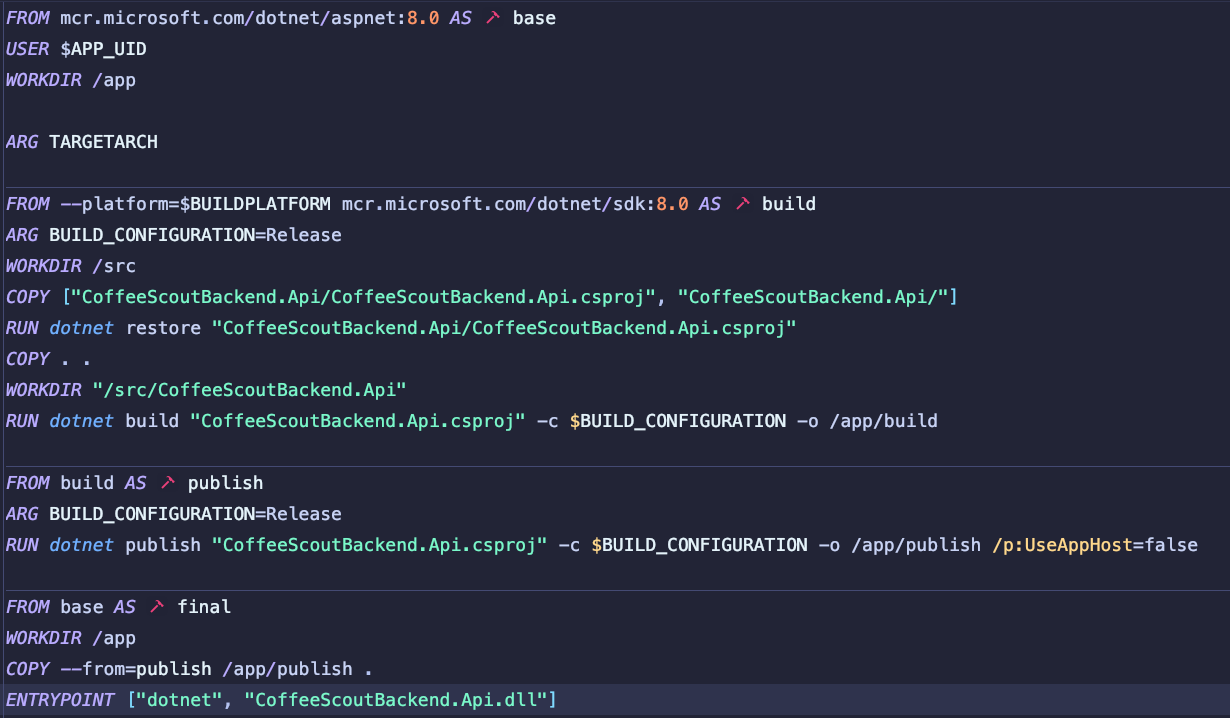
****

Рисунок 8 – Dockerfile.

Для сборки мультиплатформенного образа был использован инструмент [docker buildx](#dockerBuildx). Dev образ собирался следующей командой:

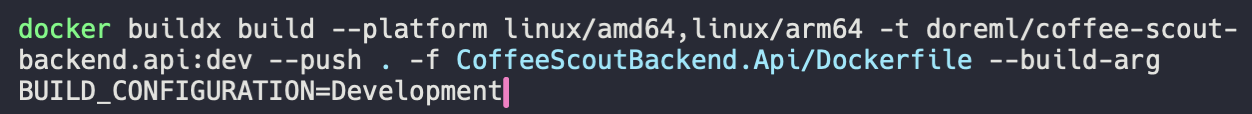


Рисунок 9 – команда для сборки образа.

В docker-compose.yml в настройках для контейнера приложения также указаны дополнительные настройки:

* **depends\_on: database** – указывает, что контейнер приложения зависит от контейнера database и должен быть запущен после него.
* **restart: always** – указывает что в случае непредвиденного завершения контейнера, он должен пробовать перезапуститься.
* **env\_file: .env** – указывает, что запуск контейнера должен быть осуществлен с использованием переменных окружения из файла .env.
* **ports: 8000:8080** – указывает, что порт 8080 контейнера должен быть связан с портом 8000 системы на которой запущен Docker. Это нужно чтобы можно было подключиться к приложению извне внутренней сети docker compose.

Так как в соответствии с бизнес-требованиями нужна была база данных Postgres с расширением [Postgis](#postgis), то было принято решение использовать образ [postgis/postgis](#postgisDocker) включающий в себя образ postgres с уже интегрированным в него расширением Postgis. Контейнер имеет следующие настройки:

* **ports: 5432:5432** - указывает, что порт 5432 контейнера должен быть связан с портом 5432 системы на которой запущен Docker. Это нужно, чтобы к базе мог подключить администратор базы данных.
* **environment:** тут указаны переменные окружения, с которыми запускается база данных.

# Описание и обоснование выбора состава технических и программных средств

# Состав технических и программных средств

Программные средства:

* Операционная система Linux с дистрибутивом Debian.
* Установленный Docker engine

Технические средства:

* 300 МБ свободного места для приложения.
* От 30 ГБ свободного места для базы данных.
* Процессор с двумя ядрами и более с частотой не менее 1 ГГц.
* 4 Гб оперативной памяти.
* Доступ к сети интернет со скорость не менее 100 мбит

# Обоснование выбора состава технических и программных средств

Так как для развертывания программы было принять решение использовать Docker, позволяющий запускать приложения в изолированной среде. Собранный Docker образ уже содержит все нужные библиотеки. Это сильно уменьшает требования к программным средствам. Linux Debian выбран, так как для базы данных используется образ postgis/postgis:16-3.4, собранный под Debian.

Требования к техническим средствам обоснованы размером Docker образа, и требованиями к скорости обработки запросов.

# Описание и обоснование выбора метода организации входных и выходных данных

# Входные данные получаются из Http запросов к Api. Либо в виде json тела запроса, либо из параметров запроса.

Выходные данные представлены в виде json ответов и Http кодов.

Это обосновано выбором REST Api.

# ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

# Ориентировочная экономическая эффективность

Данный курсовой проект не предусматривает расчет экономической эффективности.

# Предполагаемая потребность

Приложение «КофеСкаут» будет востребовано для упрощения процесса поиска и приобретения кофе с заданными параметрами и кофеен, которые могут приготовить данный напиток. Разрабатываемое приложение будет полезно как людям, которые пьют кофе каждый день, так и тем, кто хочет открыть для себя разные виды кофе и посетить новые кофейни.

# Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

На момент начала разработки посредством быстрого интернет-поиска были найдены следующие аналоги:

* Дринкит: приложение для поиска и предзаказа напитков кофейни «Дринкит», реализована большая часть функционала, однако предназначено только для одной франшизы
* Coffee map: кофейная карта, где объединено более 300 кофеен Москвы; присутствуют краткие описания для каждой кофейни; показывается расстояние до ближайших точек и маршрут
* Я хочу кофе!: сервис предзаказа еды и напитков из различных кофеен поблизости, однако работает только в Воронеже
* Coffee forest: карта кофеен с фильтрами и рейтингами; реализована большая часть функционала, однако нет возможности предзаказа
* Cofix club: приложение кофеен сети «Cofix»; есть возможность предзаказа, а также карта с кофейнями; предназначено только для одной франшизы
* OnePriceCoffee: приложение кофеен сети «OnePriceCoffee»; можно ознакомиться с ассортиментом; есть карта с кофейнями; нельзя сделать предзаказ; работает только для одной франшизы
* Stars: приложение кофеен сети «Stars Coffee»; можно ознакомиться с ассортиментом; есть карта с кофейнями; нельзя сделать предзаказ; работает только для одной франшизы

В таблице приведена сравнительная характеристика с приложениями, наиболее близкими к теме разрабатываемого проекта.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Дринкит | Coffee map | Я хочу кофе! | Coffee forest | Cofix club | OnePriceCoffee | Stars |
| Возможность предзаказа | + | - | + | - | + | - | - |
| Возможность посмотреть на карте местоположения кофеен | + | + | + | + | + | + | + |
| Возможность добавления позиций в корзину | + | - | + | - | + | + | - |
| Предлагает ближайшую кофейню с выбранным напитком | + | - | + | - | - | + | + |
| Система рекомендаций | - | - | - | + | + | + | - |
| Возможность посмотреть ассортимент выбранной кофейни | + | - | + | + | + | + | + |
| Возможность ставить оценки кофейням и напиткам | - | - | - | - | - | - | - |
| История заказов | + | - | + | - | + | + | - |
| Работает в Москве | + | + | - | + | + | + | + |
| Возможность сохранения любимых мест | - | + | + | + | + | - | - |
| Возможность сохранения любимых напитков | + | - | + | - | - | - | - |
| Построение маршрута до кофейни | + | + | - | - | - | - | + |
| Работает не только для одной франшизы или конкретной кофейни | - | + | + | + | - | - | - |

КофеСкаут представляет огромный функционал, который сделает это приложение незаменимым выбором для любого любителя кофе и просто желающего отведать вкусный напиток.

В добавок, КофеСкаут не ограничен одной кофейней, а реализует свой функционал сразу для многих кофеен.

Наличие персональных предложений сильно завлечет людей, а удобство поиска самой близкой кофейни, построения маршрута и предзаказа напитка, сильно сократит личное время пользователя, что является большим конкурентным преимуществом.

1. **ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ**
2. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
11. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
12. ГОСТ 19.401-78: Текст программы. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
13. ГОСТ 19.505-79: Руководство оператора. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
14. Спецификация OpenApi [Электронный ресурс]. URL: <https://swagger.io/specification/> (Дата обращения: 12.01.2024).
15. ASP.NET [Электронный ресурс]. URL: <https://dotnet.microsoft.com/en-us/apps/aspnet> (Дата обращения: 10.01.2024).
16. Docker образы для ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-8.0> (Дата обращения: 17.02.2024).
17. ASP.NET авторизация с ролями [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-8.0> (Дата обращения: 20.01.2024).
18. ASP.NET Core введение в Identity [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio> (Дата обращения: 15.01.2024).
19. Обзор аутентификации в ASP.NET Core [Электронный ресурс]. URL: <https://dotnet.microsoft.com/en-us/apps/aspnet> (Дата обращения: 15.01.2024).
20. Как использовать Identity для защиты Web API backend ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-8.0> (Дата обращения: 16.01.2024).
21. Безопасное хранение секретов во время разработки в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-8.0&tabs=linux> (Дата обращения: 10.01.2024).
22. Entity Framework Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-gb/ef/core/> (Дата обращения 29.01.2024).
23. Razor Pages с Entity Framework Core в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-8.0&tabs=visual-studio> (Дата обращения: 29.01.2024).
24. Введение в отношения [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/modeling/relationships> (Дата обращения: 29.01.2024).
25. Маппинг атрибутов для отношений [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/modeling/relationships/mapping-attributes> (Дата обращения: 30.01.2024).
26. Соглашения для обнаружения отношений [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/modeling/relationships/conventions> (Дата обращения: 10.01.2024).
27. Конфигурация в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-8.0> (Дата обращения: 20.01.2024).
28. Options pattern в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-8.0> (Дата обращения: 20.01.2024).
29. Dependency injection в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0> (Дата обращения: 10.01.2024).
30. ASP.NET Core Middleware [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-8.0> (Дата обращения: 15.02.2024).
31. Обработка ошибок в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-8.0> (Дата обращения: 19.02.2024).
32. Логирование в .Net и ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-8.0> (Дата обращения: 01.02.2024).
33. Http логирование в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-logging/?view=aspnetcore-8.0> (Дата обращения: 01.02.2024).
34. Routing в ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-8.0> (Дата обращения: 13.01.2024).
35. Docker [Электронный ресурс]. URL: <https://www.docker.com> (Дата обращения: 11.01.2024).
36. Docker compose [Электронный ресурс]. URL: <https://docs.docker.com/compose/> (Дата обращения: 11.01.2024).
37. Docker buildx [Электронный ресурс]. URL: <https://docs.docker.com/reference/cli/docker/buildx/> (Дата обращения: 11.01.2024).
38. Docker hub [Электронный ресурс]. URL: <https://hub.docker.com> (Дата обращения: 11.01.2024).
39. PostgresSql [Электронный ресурс]. URL: <https://www.postgresql.org> (Дата обращения: 11.01.2024).
40. Postgis [Электронный ресурс]. URL: <https://postgis.net> (Дата обращения: 11.01.2024).
41. Образ Postgis для Docker [Электронный ресурс]. URL: <https://github.com/postgis/docker-postgis> (Дата обращения: 11.01.2024).
42. Npgsql Entity Framework Core provider for PorstgresSQL [Электронный ресурс]. URL: <https://github.com/npgsql/efcore.pg> (Дата обращения: 29.01.2024).
43. FluentValidation.AspNetCore [Электронный ресурс]. URL: <https://docs.fluentvalidation.net/en/latest/> (Дата обращения: 21.01.2024).
44. SharpGrip.FluentValidation.AutoValidation.Mvc [Электронный ресурс]. URL: <https://github.com/SharpGrip/FluentValidation.AutoValidation> (Дата обращения: 21.01.2024).
45. Mapster [Электронный ресурс]. URL: <https://github.com/MapsterMapper/Mapster> (Дата обращения›4›: 01.02.2024).
46. Naming Conventions for Entity Framework Core Tables and Columns [Электронный ресурс]. URL: <https://github.com/efcore/EFCore.NamingConventions> (Дата обращения: 29.01.2024).
47. Problem details спецификация [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7807> (Дата обращения: 05.02.2024).
48. Clean Architecture в ASP.NET Core 8 [Электронный ресурс]. URL: <https://www.youtube.com/watch?v=yF9SwL0p0Y0> (Дата обращения: 10.01.2024).
49. Domain Driven Design [Электронный ресурс]. URL: <https://martinfowler.com/bliki/DomainDrivenDesign.html> (Дата обращения: 11.01.2024).
50. Что такое Entity Relation Diagram [Электронный ресурс]. URL: <https://www.lucidchart.com/pages/er-diagrams> (Дата обращения: 09.01.2024).

# ПРИЛОЖЕНИЕ 1

ТЕРМИНАЛОГИЯ

**Таблица 1.1**

|  |  |
| --- | --- |
| **Термин** | **Определение** |
| Backend | Серверная часть приложения, отвечающая за реализацию бизнес-логики. |
| REST | REpresentational State Transfer — «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. |
| API | Application programming interface - программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими. |
| REST API | API написанный в соответствии с принципами REST. |
| Endpoint | Комбинация URL и Http метода соответствующего ему. |
| Http | Cетевой протокол прикладного уровня, который изначально предназначался для получения с серверов гипертекстовых документов в формате HTML, а с течением времени стал универсальным средством взаимодействия между узлами как Всемирной паутины, так и изолированных веб-инфраструктур. |
| Архитектурный слой | Это определённый (огранниченный назначением, замкнутый) набор ресурсов (инструментов работы с ресурсами, деталей, составляющих), с помощью которых реализуются множество (огранниченное критериями) прикладных задач, характерных для данного слоя. |
| Контейнеризация | Метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Эти экземпляры (обычно называемые контейнерами или зонами) с точки зрения выполняемых в них процессов идентичны отдельному экземпляру операционной системы. |
| Bll | Business Logic Layer – слой с бизнес-логикой. |
| Dal | Data Access Layer – слой получения данных. |
| ORM | Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». |
| SQL | Structured Query Language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных. |
| Миграция | Переход от одной структуры базы данных к другой без потери консистентности. |
| Маппинг | Трансформирование одного класса в другой, посредством копирования данных. |
| Модель | Класс, представляющий сущность бизнес-логики. |
| Boilerplate код | Нетворческий программный код, который программисту приходится писать вследствие требований языка программирования, операционной системы, библиотеки подпрограмм, манеры программирования и прочего. |
| Контроллер | Класс, который отвечает за обработку запросов пользователя и управление потоком выполнения приложения |
| Сервис | Класс, который предоставляет определенные функциональные возможности или операции, доступные другим частям приложения через интерфейс. |
| Репозиторий | Класс, который предоставляет абстракцию для доступа к данным, скрывая детали реализации работы с базой данных от остальной части приложения. |
| SRID | Числовой идентификатор, который определяет конкретную пространственную систему координат (Spatial Reference System - SRS) в геоинформационных системах и пространственных базах данных. |
| Docker контейнер | Стандартизированная единица программного обеспечения, которая включает в себя все необходимые компоненты для запуска приложения: код, библиотеки, среду выполнения и системные инструменты. |
| Docker образ | Файловая система, которая содержит все необходимые компоненты для запуска приложения в контейнере Docker. Образ включает в себя исполняемый код, библиотеки, зависимости, настройки и другие файлы, необходимые для работы приложения в изолированной среде. |
| Swager | Инструмент, который позволяет автоматически, документировать API. |
| Dockerfile | Текстовый файл, который содержит инструкции по созданию Docker образа. |
| Порт | Числовой идентификатор, который используется для определения конкретного процесса или сервиса, с которым другие устройства могут обмениваться данными. |
| Переменная окружения | Динамическая переменная, которая определяется в операционной системе и доступна для всех процессов, работающих в данной среде. Они используются для хранения информации о конфигурации и настройках системы или приложения, а также для передачи данных между процессами. |
| Docker engine | Центральная компонента Docker, которая обеспечивает основную функциональность контейнеризации: управление контейнерами, управление образами, сетями и томами, а также другие операции, связанные с контейнерами. |
| Дистрибутив | Операционная система Linux, которая поставляется с определенным набором программного обеспечения и инструментов, предварительно настроенных и упакованных вместе. Это версия Linux, которая предоставляется конечному пользователю в виде готовой к использованию операционной системы. |

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |