# Введение

Тема проекта – интернет-магазин брендовой обуви с использованием базы данных.

С развитием технологий и повсеместным доступом к интернету, онлайн-торговля становится все более популярной. Особенно это заметно в сегменте продажи обуви, в частности кроссовок, которые являются неотъемлемой частью гардероба многих людей. Несмотря на множество существующих платформ, многие из них сталкиваются с рядом проблем, влияющих на удобство использования, производительность и безопасность.

Целью данного курсового проекта является разработка веб-приложения интернет-магазина брендовой обуви с применением архитектуры микросервисов. Основное внимание уделяется созданию удобного и функционального решения, которое отвечает современным требованиям к веб-приложениям, включая масштабируемость, модульность и простоту сопровождения. Разработка будет осуществляться с использованием C# ASP.Net Core на стороне сервера и TypeScript React на стороне клиента.

Архитектура микросервисов выбрана для обеспечения гибкости и упрощения управления приложением. Разделение функциональности на отдельные микросервисы позволяет упростить внедрение новых функций, минимизировать взаимное влияние компонентов и обеспечить высокую отказоустойчивость.

# ГЛАВА 1. Аналитическая часть

# 1.1 Анализ предметной области

Электронная коммерция, или e-commerce, представляет собой систему продажи товаров и услуг через интернет. Основным преимуществом интернет-магазинов является удобство для пользователей, возможность быстро находить и приобретать нужные товары, сравнивать их характеристики и получать их с доставкой на дом.

Сегмент брендовой обуви занимает важное место на рынке e-commerce. Покупатели ценят высококачественные товары известных брендов, и удобство выбора таких товаров в онлайн-магазине становится ключевым фактором успеха. Для эффективной работы интернет-магазина брендовой обуви необходимы:

* удобный интерфейс для пользователей
* безопасная авторизация и работа с корзиной
* система управления товарами

# 1.2 Обоснование необходимости создания проекта

На данный момент наблюдается высокая конкуренция в сегменте интернет-магазинов. Тем не менее, значительное количество существующих решений обладает недостатками:

* Сложный и неудобный пользовательский интерфейс
* Низкая производительность при большом количестве пользователей
* Ограниченные возможности по управлению каталогом товаров
* Отсутствие гибкости в модернизации системы и внедрении новых функций

Проектируемое приложение нацелено на решение этих проблем за счет использования современной архитектуры и технологий:

Применение микросервисной архитектуры обеспечивает масштабируемость и упрощает поддержку приложения.

Использование C# ASP.Net Core на серверной стороне гарантирует надежность и производительность.

Внедрение TypeScript React на клиентской стороне позволяет создать динамичный и удобный интерфейс.

# 1.3 Обзор существующих решений

На рынке представлены следующие типы решений для интернет-магазинов:

1. Готовые платформы (Poizon, StockX): предоставляют широкий функционал, но ограничивают кастомизацию под специфические бизнес-потребности.
2. Кастомные разработки: обеспечивают полную гибкость, но требуют значительных временных и финансовых затрат.
3. Решения с использованием микросервисов: преимущественно реализуются крупными компаниями для обеспечения высокой надежности и масштабируемости.

Однако большинство существующих решений либо недостаточно гибки, либо сложны в реализации для среднего и малого бизнеса. Разработка нового интернет-магазина с акцентом на брендовую обувь позволяет создать продукт, идеально соответствующий потребностям пользователя.

# 1.4 Выявление проблем и ограничений

Основные проблемы, которые могут возникнуть при реализации проекта:

1. Сложность интеграции микросервисов: требует четкого планирования взаимодействия между ними.
2. Обеспечение безопасности: необходимо защитить данные пользователей.
3. Удобство пользовательского интерфейса: важна адаптивность и интуитивность интерфейса для пользователей с разным уровнем технической подготовки.

# 1.5 Основные компоненты системы

Проектируемая система будет включать следующие основные компоненты:

1. **Микросервис авторизации**: отвечает за регистрацию, вход в систему и управление профилями пользователей.
2. **Микросервис управления товарами**: обрабатывает запросы, связанные с каталогом товаров, включая добавление новых позиций, редактирование характеристик и цен.
3. **Frontend**: интерфейс пользователя, реализованный на React. Сюда входит:
   * + Меню с кнопками авторизации и корзины.
     + Каталог товаров с карточками, содержащими описание и возможность добавления в корзину.
     + Страница карточки товара для детального описания.
4. **Микросервис оформления заказа**: функциональность для добавления товаров, их просмотра и оформления заказа.

Ниже представлены взаимодействия между компонентами интернет-магазина, основанного на микросервисной архитектуре:

1. Пользовательский запрос:

* Пользователь (через браузер) взаимодействует с фронтендом.
* Например, открывает каталог или вводит данные для входа.

1. Frontend отправляет запрос на сервер:

* Запрос отправляется в виде HTTP-запроса (REST API) к серверу (backend).

1. Backend маршрутизирует запрос:

* Сервер анализирует запрос и перенаправляет его в соответствующий микросервис:
* Авторизация → Микросервис авторизации.
* Просмотр товаров → Микросервис управления товарами.

1. Микросервисы обрабатывают запрос:

* Если запрос требует доступа к данным, микросервис обращается к базе данных.
* Результат обработки возвращается через backend к фронтенду.

1. Ответ пользователю:

* Фронтенд получает данные (например, список товаров) и обновляет интерфейс.

Ниже представлена схема будущей базы данных:

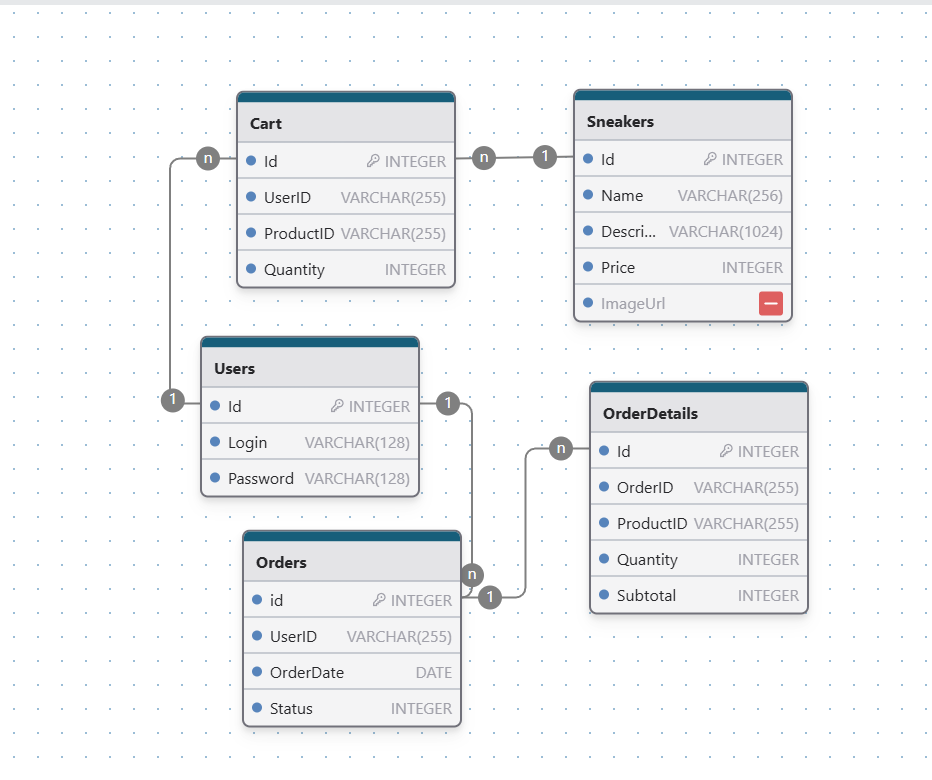


Рисунок 2.1. Таблица Users в базе данных.

# ГЛАВА 2. Разработка веб-сайта

# 2.1 Проектирование и физическая реализация базы данных на СУБД

Для реализации данного проекта была создана база данных. Таблица реляционной базы данных — это совокупность связанных данных, хранящихся в структурированном виде в базе данных. На рисунках ниже представлены эти таблицы в PostgeSQL.

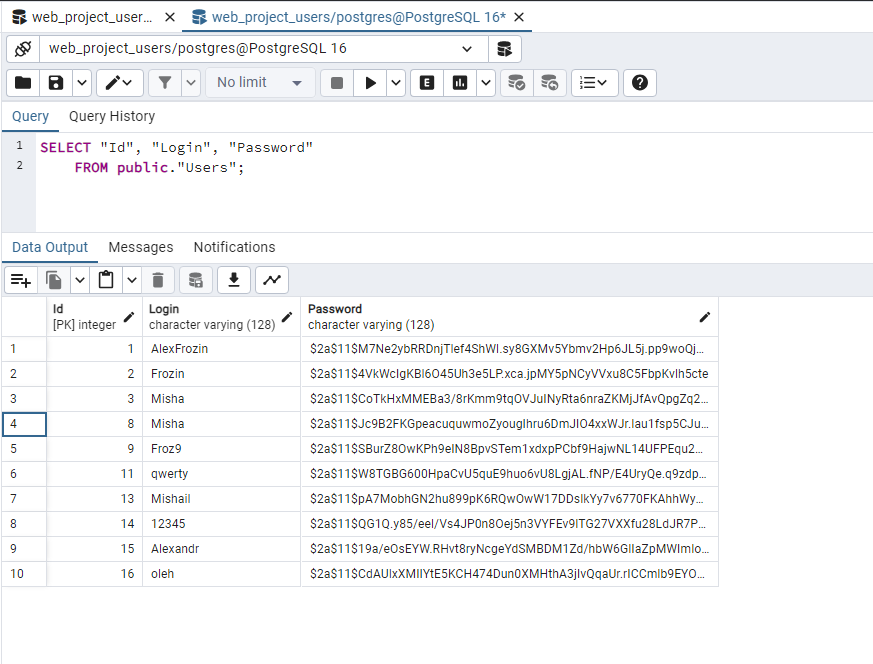


Рисунок 2.1. Таблица Users в базе данных.

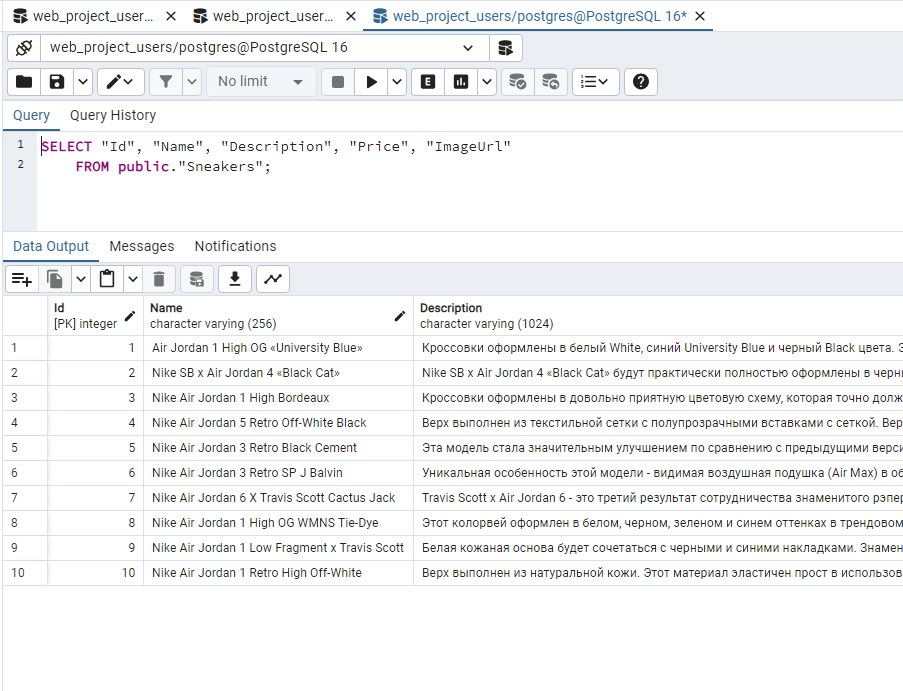


Рисунок 2.2. Таблица Sneakers в базе данных.

Таблица Users состоит из трех элементов:

* Id
* Login
* Password

Таблица Sneakers состоит из пяти элементов:

* Id
* Name
* Description
* Price
* ImageURL

Так, с помощью PostgreSQL была успешно спроектирована и реализована база данных для будущего веб-сайта.

# 2.2 Подключение базы данных к коду

Чтобы понимать, как данный проект был реализован, необходимо рассмотреть основные части программного кода. Далее представлены главные функции приложения в программном виде.

Для начала рассмотрим реализацию подключения к базе данных Backend части кода. Для этого был создан классs User, Sneaker, Dbcontext. Ниже представлены листинги этих классов:

public class User

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(128)] public string Login { get; set; }

[MaxLength(128)] public string Password { get; set; }

}

Листинг 2.1. Код класса User.

public class Sneaker

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(256)] public string Name { get; set; }

[MaxLength(1024)] public string Description { get; set; }

public int Price { get; set; }

[MaxLength(1024)] public string ImageUrl { get; set; }

}

Листинг 2.2. Код класса Sneaker.

public class AppDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Sneaker> Sneakers { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<User>()

.HasKey(h => new { h.Id });

modelBuilder.Entity<Sneaker>()

.HasKey(h => new { h.Id });

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=web\_project\_users;Username=postgres;Password=1234");

}

}

Листинг 2.3. Код класса DbContext.

Именно в классе DbContext прописана строка для подключения кода к базе данных, чтобы в будущем с ней работать при создании нашего веб-сайта.

# 2.3 Разработка бэкенда

Настройка проекта в файле Program.cs выглядит следующим образом:

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Backend\_SneakerShop.Data;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddCors(options =>

{

options.AddDefaultPolicy(builder =>

{

builder.WithOrigins("http://localhost:5173")

.AllowAnyHeader()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseRouting();

app.UseCors();

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

Листинг 2.4. Настройка проекта ASP.NET Core.

Для создания авторизации был создан контроллер AuthController:

[ApiController]

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

public class AuthController : ControllerBase

{

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] UserDto userDto)

{

if (userDto.Password.Length < 6 || !userDto.Password.Any(char.IsDigit) || !userDto.Password.Any(char.IsLetter))

{

return BadRequest("Password must be at least 6 characters long and contain both letters and numbers.");

}

var user = new User

{

Login = userDto.Login,

Password = BCrypt.Net.BCrypt.HashPassword(userDto.Password)

};

var DbContext\_user = new AppDbContext();

DbContext\_user.Users.Add(user);

DbContext\_user.SaveChanges();

return Ok();

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] UserDto userDto)

{

try

{

var DbContext\_user = new AppDbContext();

var user = (from u in DbContext\_user.Users

where u.Login == userDto.Login

select u).FirstOrDefault();

if (user == null || !BCrypt.Net.BCrypt.Verify(userDto.Password, user.Password))

{

return Unauthorized();

}

return Ok();

}

catch (Exception ex)

{

return StatusCode(500, "Internal server error");

}

}

Листинг 2.5. Контроллер для работы с авторизацией.

Контроллер AuthController предоставляет API для регистрации новых пользователей и авторизации уже зарегистрированных пользователей. Он реализует проверку корректности вводимых данных и обеспечивает безопасность хранения паролей с помощью хэширования.

1. Атрибуты контроллера
   * [ApiController]: указывает, что данный контроллер является API-контроллером. Это автоматически добавляет некоторые полезные функции, такие как автоматическая проверка модели.
   * [Route("api/[controller]")]: устанавливает базовый маршрут для всех методов контроллера. В данном случае маршрут будет api/Auth.
2. Метод Register
   * [HttpPost("register")]: Атрибут, указывающий, что данный метод обрабатывает HTTP POST запросы по маршруту api/Auth/register.
   * Register([FromBody] UserDto userDto): Метод принимает данные пользователя (логин и пароль) в виде объекта UserDto.
   * Проверка пароля: если длина пароля меньше 6 символов или он не содержит цифр и букв, возвращается ошибка BadRequest с сообщением о некорректном пароле.
   * Создание пользователя: если проверка прошла успешно, создается объект User с логином и хэшированным паролем.
   * Сохранение пользователя: Пользователь сохраняется в базе данных.
   * Возврат результата: в случае успешной регистрации возвращается статус Ok.
3. Метод Login
   * [HttpPost("login")]: Атрибут, указывающий, что данный метод обрабатывает HTTP POST запросы по маршруту api/Auth/login.
   * Login([FromBody] UserDto userDto): Метод принимает данные для входа (логин и пароль) в виде объекта UserDto.
   * Поиск пользователя: Метод ищет пользователя в базе данных по логину.
   * Проверка пароля: если пользователь не найден или пароль не совпадает, возвращается статус Unauthorized.
   * Возврат результата: в случае успешной авторизации возвращается статус Ok. В случае ошибки сервера возвращается статус Internal Server Error.

Применение технологий

1. BCrypt: Библиотека для хэширования паролей, обеспечивающая безопасность их хранения.
2. Entity Framework: ORM для работы с базой данных, используемая для управления пользователями в базе данных.

Контроллер AuthController обеспечивает функциональность регистрации и авторизации пользователей. Он включает в себя методы для создания новых пользователей с проверкой пароля и безопасным хранением паролей, а также метод для проверки логина и пароля при входе в систему. Этот контроллер является важной частью обеспечения безопасности и удобства использования интернет-магазина.

Для работы с базой данных и корректного отображения информации о кроссовках в карточках товаров был создан контроллер SneakerController.cs:

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

[ApiController]

public class SneakersController : ControllerBase

{

[HttpGet]

public ActionResult<IEnumerable<Sneaker>> GetSneakers()

{

var DbContext\_sneaker = new AppDbContext();

var Sneakers = from sneaker in DbContext\_sneaker.Sneakers select sneaker;

return Ok(Sneakers);

}

}

Листинг 2.6. Контроллер для отображения информации в карточках товаров.

Контроллер SneakersController предоставляет API для получения списка всех кроссовок, хранящихся в базе данных. Он используется для отображения товаров на клиентской стороне интернет-магазина.

1. Атрибуты контроллера
   * [Route("api/[controller]/[action]")]: устанавливает маршрут для всех методов контроллера, включающий имя контроллера и имя действия. В данном случае маршрут будет api/Sneakers/GetSneakers.
   * [ApiController]: указывает, что данный контроллер является API-контроллером. Это автоматически добавляет некоторые полезные функции, такие как автоматическая проверка модели.
2. Метод GetSneakers
   * [HttpGet]: Атрибут, указывающий, что данный метод обрабатывает HTTP GET запросы. Метод будет доступен по маршруту api/Sneakers/GetSneakers.
   * ActionResult<IEnumerable<Sneaker>> GetSneakers(): Метод возвращает список всех кроссовок в базе данных.
   * Создание контекста базы данных: создаётся экземпляр класса AppDbContext для взаимодействия с базой данных.
   * Получение списка кроссовок: выполняется LINQ-запрос для получения всех объектов Sneaker из таблицы Sneakers.
   * Возврат результата: возвращается статус Ok с полученным списком кроссовок.

Применение технологий

1. Entity Framework: ORM для работы с базой данных, используемая для управления данными о кроссовках.
2. LINQ: Язык запросов для работы с коллекциями данных, используемый для выборки данных из базы данных.

Контроллер SneakersController обеспечивает функциональность получения данных о товарах (кроссовках) из базы данных. Он включает в себя метод для выборки всех кроссовок и возвращает их в формате JSON, что позволяет легко интегрировать эти данные с клиентской частью интернет-магазина. Этот контроллер является важной частью архитектуры сайта, обеспечивая доступ к данным о товарах.

# 2.5 Тестирование веб-сайта