Министерство образования и молодежной политики
Свердловской области

ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Специальность 09.02.07: «Информационные системы и программирование»

Разработка автоматизированной системы для кафе

Пояснительная записка

к курсовому проекту

КП-ПР-41-21-2026-ПЗ

Разработал:

Студент гр. ПР-41 _____/ М.К. Шмелев

Руководитель _____/ Д.О. Гариев

2026

Министерство образования и молодежной политики
Свердловской области

ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Специальность 09.02.07: «Информационные системы и программирование»

Разработка автоматизированной системы для кафе

Курсовой проект

КП-ПР-41-21-2026-ПЗ

2026

Оглавление

Введение

Информационные технологии сегодня являются ключевым фактором в нашей жизни и на каждом предприятии. Сфера услуг общественного питания также нуждается в автоматизации рутинных процессов, связанных с заказами. Компании ищут лучшие автоматизированные системы из-за высокой конкуренции за сотрудников и клиентов.

Актуальность темы подтверждается тем, что каждый день создаются новые организации, которым требуются такие системы. На рынке много предложений: есть как дорогие решения, так и попроще для менее популярных заведений. Всё зависит от требований заказчика.

Целью курсового проекта является разработка автоматизированной системы, состоящей из серверной части и клиентского приложения, для автоматизации обработки заказов в кафе.

Задачи, которые необходимо выполнить для достижения поставленной цели:

1. Выполнить анализ предметной области.
2. Провести обзор существующих аналогов и показать преимущества собственной разработки.
3. Реализовать базу данных и логическую структуру приложения.
4. Реализовать серверную часть приложения.
5. Реализовать клиентскую часть приложения.
6. Разработать пользовательский интерфейс.
7. Осуществить тестирование разработанных модулей.

1 Постановка задачи

1.1 Описание предметной области

Предметной областью является управление предприятием общественного питания.

В настоящее время, если в заведении нет автоматизированной системы, процесс выглядит так: официант принимает заказ, записывает его в блокнот, затем передает данные кассиру и повару.

Этот подход очень неудобен, если в зале более пятидесяти столов и все они заняты, а официантов всего один или два человека, они физически не успеют качественно обслужить всех гостей. При высокой нагрузке смысл ручного метода пропадает.

Отсутствие контроля провоцирует целый ряд ошибок – блюда часто выносятся не на те столы, что вызывает раздражение гостей и конфликт с персоналом.

Отдельная проблема - нарушение курсов подачи. Официант может вынести десерт раньше основного блюда, что испортит впечатление от вкуса и от вечера. Кроме того, важна одновременная подача: если гости заказали два блюда, недопустимо принести одно блюдо, заставив второго гостя ждать и смотреть, как первый ест. Диаграмма, изображённая на рисунке *1,* отображает данный подход.

Также существует риск кражи: сотрудники могут обманывать руководство, присваивая разницу между чеком и полученными деньгами. В разрабатываемой системе происходит учет денежных средств, что обеспечивает отчетность. Из этого можно сделать вывод: в XXI веке ни одно предприятие общественного питания не сможет выдержать конкуренцию и работать без автоматизированной системы.

Диаграмма показывает протекающий процесс актеров без автоматизированной системы: гость заказывает позицию, оплачивает, официант записывает в блокнот, передает данные на кассу, касса печатает чек официанту на оплату. Повар готовит заказ и отдает официанту, демонстрация на рисунке 1.

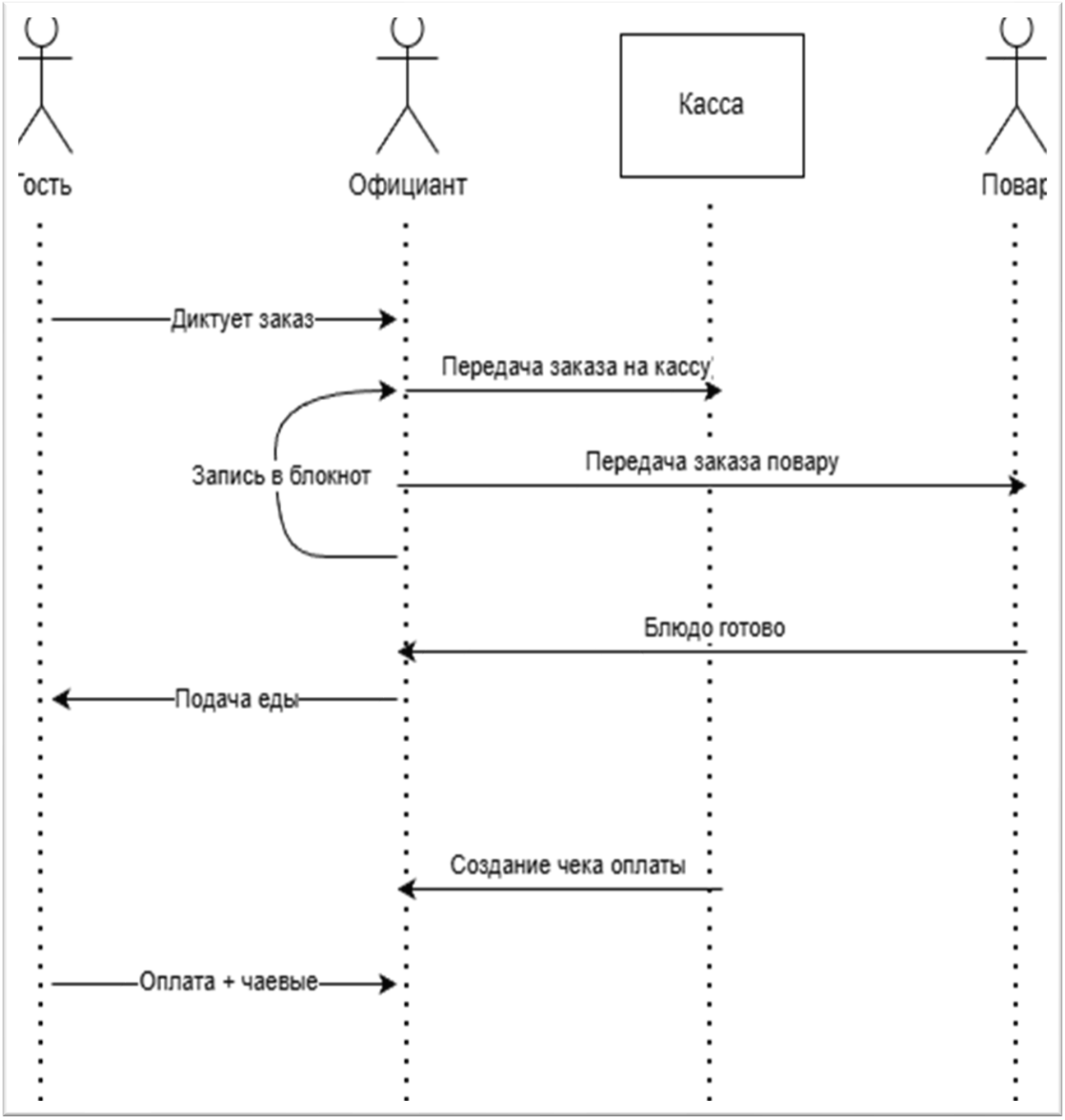


Рисунок 1 – Диаграмма последовательности.

1.2 Требования к программному продукту

Проектируемая система должна выполнять следующие задачи:

1. Авторизация и доступ к функционалу по ролям: система должна авторизовать пользователя (администратор, официант, повар).
2. Управление меню: администратор имеет возможность редактировать, добавлять, удалять позиции меню, менять стоимость, категории и описания.
3. Приём заказов: официант через приложение создает заказ, привязывает его к столу и добавляет блюда, а также может менять заказ.
4. Статусы заказов: администратор или же официант могут его изменять, но если заказ попал в систему автоматически, то он будет иметь статус завершен.
5. Вывод всех позиций: как для официанта, так и для повара вывод позиций, которые нужно отдавать.
6. Оплата: автоматический расчет стоимости.

Сравнение программного обеспечения с его аналогами***:*** на таблице 1 проведено сравнение аналогов.

1. IIKO – предоставление комплексной автоматизированной системы, решение для всех типов заведений. Охватывает все аспекты бизнеса – от кассы до склада. Недостаток в том, что имеет высокую стоимость для малого бизнеса от 10 - 32 тыс. руб.
2. R-Keeper – для сложных и очень крупных сетей. Архитектура и зависимость к оборудованию, плюсом нужен обученный человек чтобы постоянно следить за системой. Огромное количество разных модулей продаж. Дорогая система – для покупки полной версии приложения как от 60 – 150 тыс. руб.

Разрабатываемая система – менее нагруженная, бесплатная, не требует мощного ПО, можно обойтись без POS–терминалов. Запутаться в функциях не получится, есть базовые и готовые решения для обработки заказов. Это экономит время обучение персонала, сложность минимальная. Низкие системные требования – этот продукт можно развернуть на любом офисном пк. И заказчик получает

полный контроль над сервером, что в будущем играет большую роль в расширении и масштабировании.

Таблица 1 - сравнение лидеров рынка с разрабатываемой системой

Продукты:	*IIKO*	*R-Keeper*	*Разрабатываемая система*
аудитория	Все типы заведений	крупные ресторанные сети.	*Малый и средний бизнес*.
Стоимость внедрения	Средняя варьируется от 10.000 – 32.000 тыс.руб. еще ежемесячная подписка	Высокая (60 000 – 150 000 руб. за полную версию).	*Бесплатная* (затраты только на разработку и внедрение).
Оборудование	Требуются POS-терминалы и совместимое оборудование.	Высокие требования к ПК и POS - Терминалам	*Низкие требования*. Работает на любом офисном ПК,
Сложность обучение	Охватывает все аспекты (от кассы до склада), требует обучения.	Высокая сложность. Очень много модулей ведения учета.	*Минимальная сложность*. Быстрое обучения персонала.
Обслуживание	Требует техподдержки.	Требуется обученный специалист.	*Простое*. Не требует обученного специалиста.
Функциональность	Комплексная (Касса, Склад, Кухня, Финансы, Персонал).	Мощная система с разными вариантами продаж.	*Готовое решения* для обработки заказов, ничего лишнего.

Чтобы все задачи были реализованы правильно, к проекту предъявляются следующие *функциональные требования:* операции над пользователем, аутентификация пользователей. Система требует ввод логина и пароля. Регистрация пользователя: требует ФИО, логин, пароль. Выбор роли: только через права администратора. Также вывод пользователей.

Операции над заказом: создание заказа, система требует выбрать позицию из списка меню, и привязывает заказ к определенному столу. Редактирование заказа: возможность удаления и добавления позиций в заказ, количество порций. Автоматический расчет стоимости: система мгновенно рассчитывает стоимость заказа при любом изменении. Отправка повару: передача данных на кухню, вывод позиций для готовки. Закрытие: когда гости готовы оплатить официант закрывает заказ и предлагает оплатить счет, смена статуса заказа “Оплачен”, “Готов” смена зависит от действий с заказом.

Управление меню: просмотр и редактирование позиций, возможность добавления и удаления. Управление статусом: показывает в стоп-листе ли блюдо. Ценообразование: изменение стоимости блюд.

Кухня: подробный вывод заказов и детализация позиций (номер стола, название блюда, количество), контроль готовности.

2 Проектирование

Проектирование важная часть в разработке программного обеспечения (ПО) Начнем с того, какие требования для проекта автоматизированной системы для кафе:

Стек технологий сервер:

1. Бекэнд: ASP.NET Core API, .NET 9. C#, Слоистая архитектура.
2. Представление (UI): WPF (приложение), .NET 9.
3. База данных: SQL Server и объектно-реляционный преобразователь (ORM) Entity Framework.
4. Безопасность: JWT (JSON Web Token), BCrypt – Шифрование пароля.
5. Документация API: Swagger.

Стек технологий клиент:

1. Бекэнд: .NET 9, C#.
2. Представление (UI): WPF (приложение), .NET 9.
3. Паттерн: Code-Behind (Event-driven).
4. Взаимодействие с API: HttpClient, Text.Json.
5. Дизайн: MaterialDesignInXamlToolkit.

2.1 Проектирование логической структуры

Логическая структура построена на основе модели баз данных. Главная задача была сделать правильную структуру, следуя современным практикам, чтобы обеспечить модульность и возможность масштабирования. Для понимания представлена ER Диаграмма базы данных, изображенная на рисунке 2. Которая служит основой для проектирования базовых классов сущностей. Она показывает название полей, связи сущностей, ключи, типы данных. Это очень важный пункт при проектировании логической структуры.

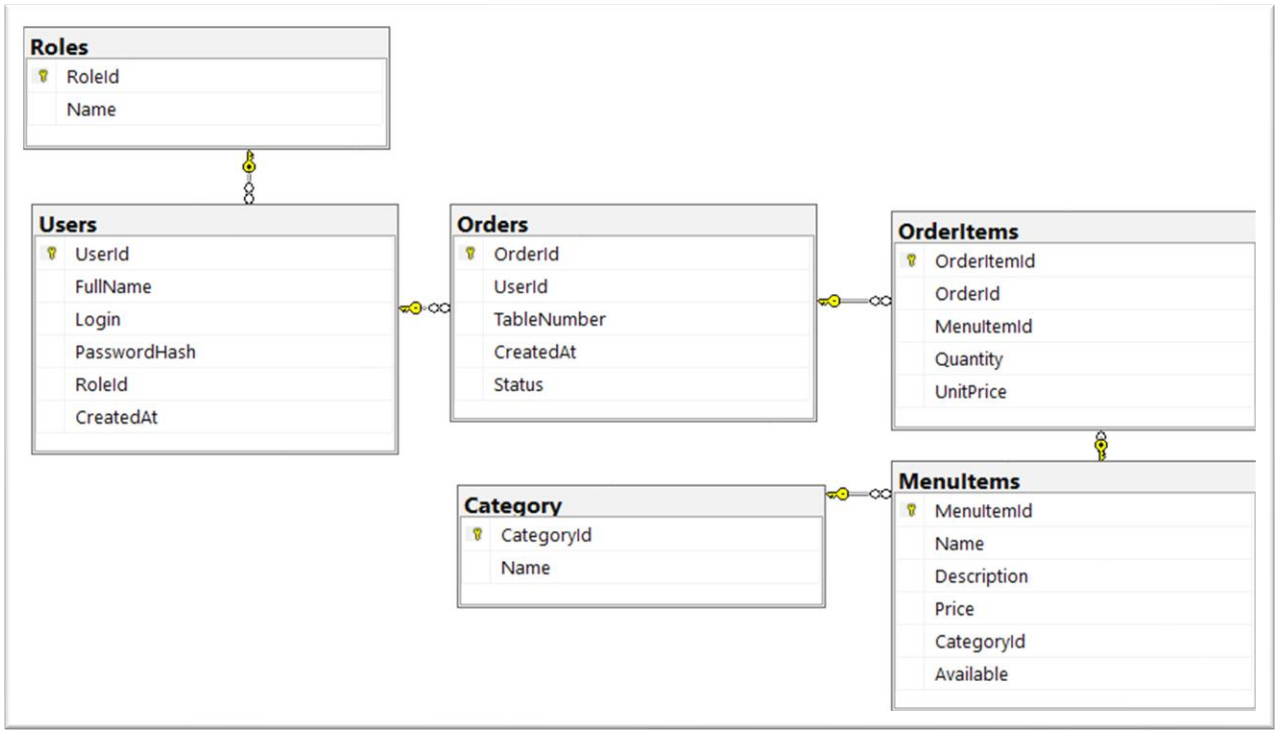


Рисунок 2 - ER Диаграмма

Основными сущностями являются: User, Order, MenuItem, OrderItem, вот что они из себя представляют:

User - ключ таблицы UserId. Остальные поля: Login, PasswordHash, FullName. Имеет связь многие к одному с сущностью Role.

Order (Заказ) - главная сущность, отражающая обслуживание клиента. Ключ OrderId. Остальные поля: TableNumber, Status, CreatedAt, UserId. Связи: связана с User (официант, открывший заказ) и содержит список OrderItems.

MenuItem - справочник блюд и напитков. Ключ MenuItemId. Остальные поля: Name, Description, Price, Available, CategoryId. Связи: относится к одной Category.

OrderItems - позиции заказа. Ключ OrderItemId. Остальные поля: OrderId, MenuItemId, Quantity, UnitPrice.

Category - категории блюд. Ключ CategoryId, Остальное Name.

UML - диаграммы: диаграмма последовательности показывает, как протекает бизнес-процесс во время его работы, как работают сотрудники, система, представлено на рисунке 3. На диаграмме изображена последовательность действий при выборе автоматизированной системы для обработки заказов.

Клиент делает заказ у официанта, при необходимости может изменить выбранные позиции и затем производит оплату. Официант принимает заказ и вносит его в систему. Повару предоставляется интерфейс для просмотра поступивших заказов и их статусов. WPF-приложение выступает связующим звеном между персоналом и сервером: все действия сотрудников передаются на сервер через приложение. Серверное API содержит основную бизнес-логику и отвечает за обработку заказов.

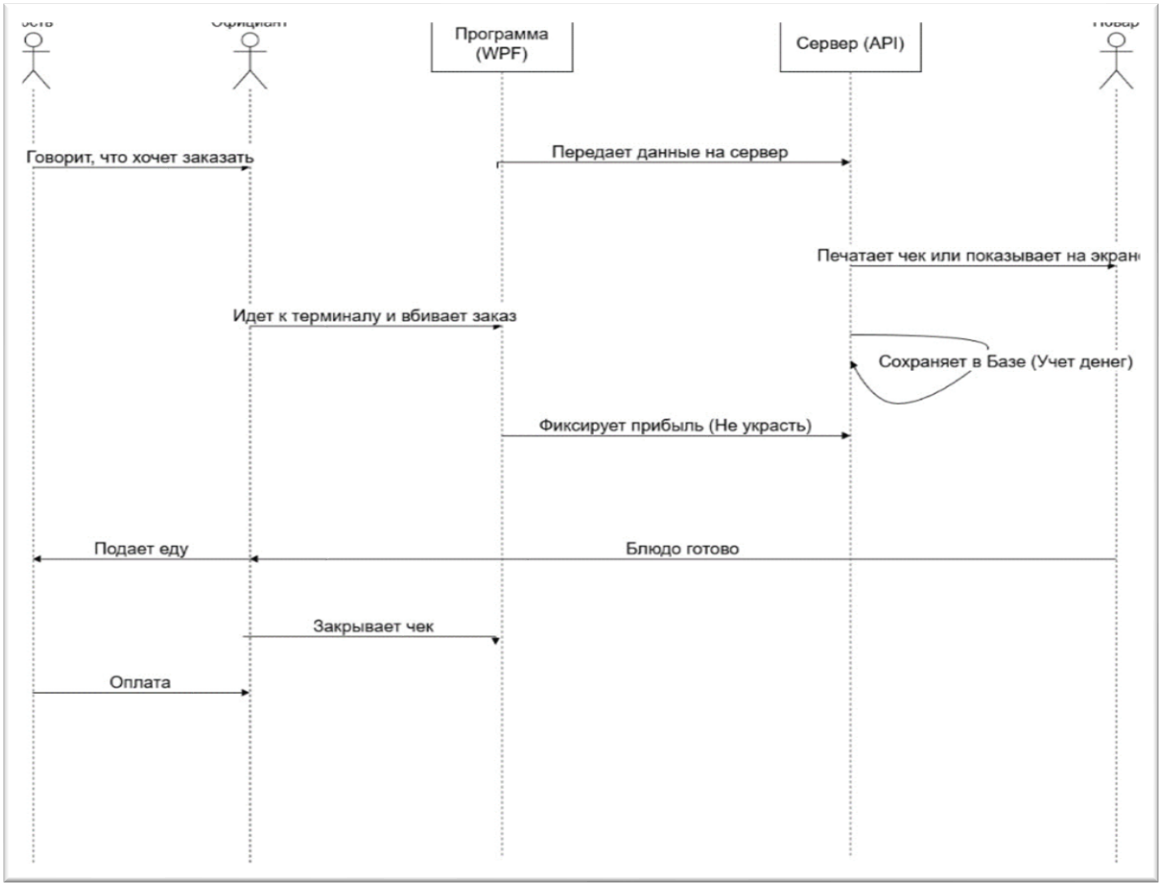


Рисунок 3 - Диаграмма последовательности с использованием системы

Процесс взаимодействия можно описать таким: официант принимает заказ у клиента на стойке с POS - терминалом или же блокнотом, подходит к стойке, заводит заказ в систему в клиентском приложении, дальше данные валидируются и отправляются по API на обработку на сервер с ними происходит прописанная логика, различные проверки и сервер возвращает ответ, дальше заказ переходит на кухню к поварам, когда приготовят официант отмечает в клиент приложении что заказ готов, заказ уходит на сервер и отмечается как готов и официант его

приносит. Дополнительно система обеспечивает синхронизацию статусов заказов между клиентским приложением и серверной частью в реальном времени.

На рисунке 4 изображена диаграмма вариантов использования (прецедентов). Диаграмма показывает актеров, которые оперируют с автоматизированной системой. Каждый актер делает свое дело, официант и повар вместе могут смотреть товары, менеджер управляет меню, учетными записями. Это диаграмма показывает процесс взаимодействия прецедентов.

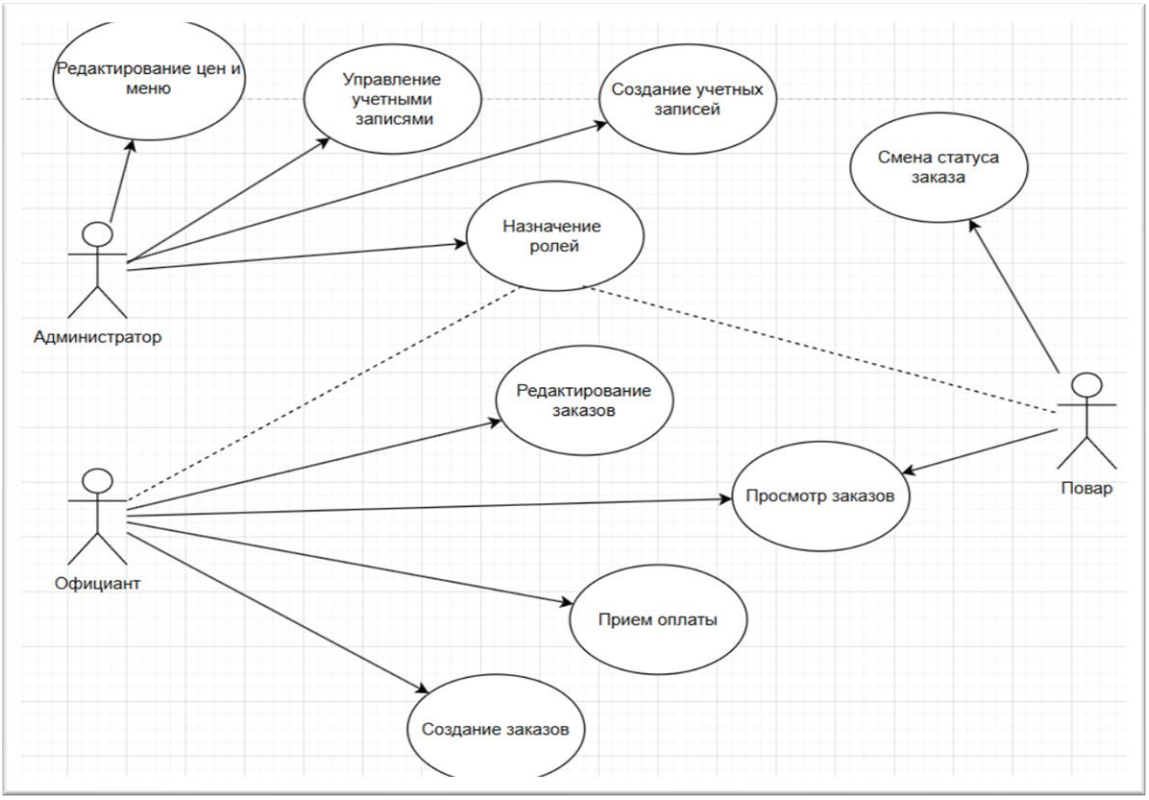


Рисунок 4 - Диаграмма прецедентов

2.2 Проектирование физической структуры

Физическая структура продукта реализована в соответствии с паттерном архитектуры N-Layer (Многослойная архитектура) на основе MVC (Model View Controller). На платформе .NET 9. Проект будет организован в виде решения CafeSolution. Файловая структура проекта выглядит следующим образом.

1. Controllers: (OrdersController.cs, MenuController.cs, UserController.cs) – слой представления, отвечает за прием HTTP запросов от клиента (WPF), валидация данных на входе, отвечает и принимает данные в формате Json.

2. Services: (OrderService.cs, UserService.cs, TokenService) – слой бизнес логики. Здесь происходят вычисления заказов, управление пользователями, управление транзакциями.

3. Repositories - отдельный слой, который работает с базой. Предоставляет базовые методы CRUD с моделями.

4. Models - модели, которые напрямую связаны с таблицами базы данных.

5. DTOs - объекты для передачи данных между клиентом и сервером. Разделены по папкам (MenuItems, OrderItems, Orders, Roles, Users, Category, Kitchen).

6. Data (CafeDbContext) – контекст базы данных, отвечает за соединение с СУБД.

В качестве выбранной базы данных будет использоваться MS SQL и будет управляться через SQL SERVER. Взаимодействие с БД будет использована библиотека EF-Core (Entity Framework). Классы, размещённые в папке Models, соответствуют сущностям базы данных. Миграции базы данных хранятся в каталоге Migrations.

Чтобы понимать, как работает система, а именно паттерн, ниже представлена схема на рисунке 5. На изображении файловое дерево проекта, показано какой паттерн проектирования использован и в какой последовательности взаимодействует каждый модуль паттерна.

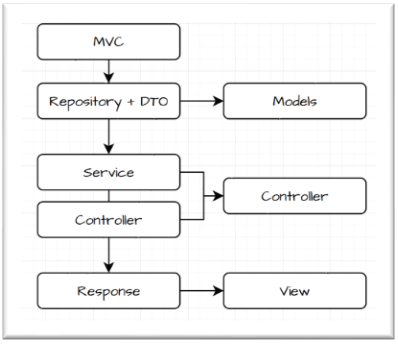


Рисунок 5 - Файловое дерево проекта

Для документирования API – конечных точек для клиента будет использоваться Swagger. У HTTP есть свои методы: GET – получить, POST – создать, PUT – обновить, DELETE – удалить, PATCH – частичное обновление.

Ниже документирование конечных точек API с использованием инструмента Swagger. Дополнительно ссылка на опубликованную спецификацию API и QR-код приведены в приложении Б

Menu – здесь конечные точки категории menu, а именно получение (GetMenu), добавление (Add), обновление (Update), удаление (DeleteItem), получение по id (MenuItemId). Права администратора требуются: Add, Update, DeleteItem, MenuItemId. Все это представлено на рисунке 6.

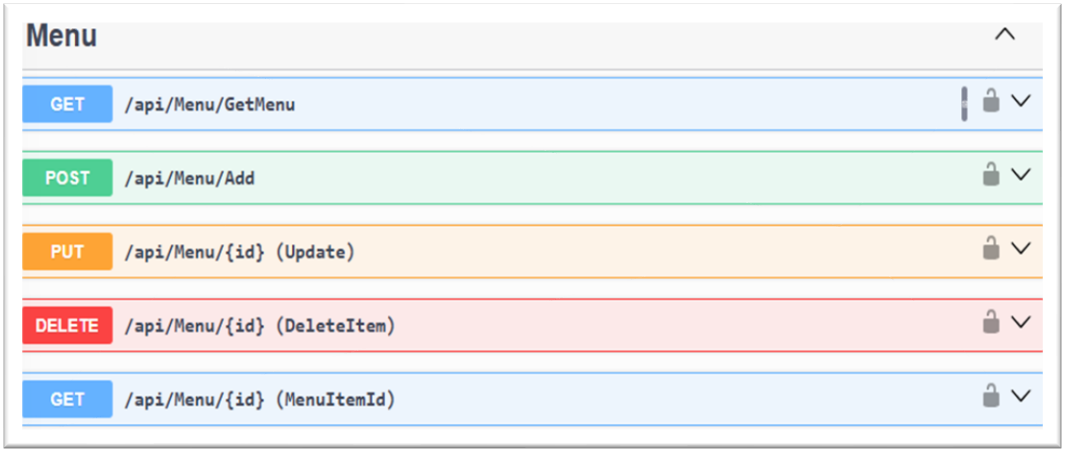


Рисунок 6 - Конечные точки Menu

Для запросов по адресу *Menu* используется DTO: MenuItemResponseDto – возвращает ответ после запроса к конечной точке, сам класс выбирает что возвращать в ответ вместо полного готового объекта. CreateMenuItemDto – когда идет обращение к конечной точке то данный класс реализует макет данных, который должен прийти. UpdateMenuItemDto – когда выбран объект, который нужно обновить используется этот класс он не позволяет обновлять все поля, а только которые в нем прописаны.

Orders – здесь конечные точки категории orders, а именно создание заказа, получение всех заказов, получение по id, добавить позиции в заказ, удалить заказ, получить заказ привязанный к user, получить статус заказа, обновить статус заказа, и удалить позицию из заказа, все это представлено на рисунке 7.

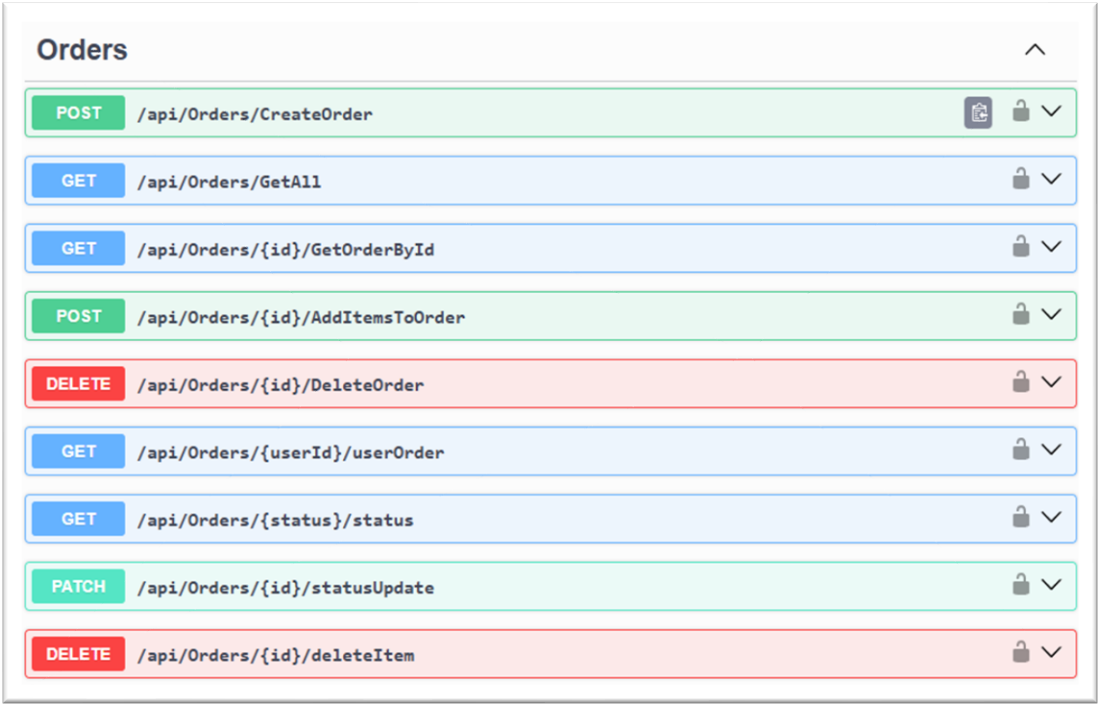


Рисунок 7 - Конечные точки

Для запросов по адресу *Orders* используется DTO OrderResponseDto – возвращает полный ответ сервера после создания или запроса заказа, содержит статус, дату создания и вложенный список блюд, скрывая служебную логику. CreateOrderDto – реализует макет данных при создании нового заказа, требует указать сотрудника, номер стола и список позиций. OrderItemDto – используется как вложенный объект, описывает конкретное блюдо и его количество в составе заказа.

User – модуль, содержащий конечные точки для управления пользователями системы: вход в приложение (Login), регистрация (Register), получение списка пользователей (User), получение пользователя по идентификатору (GetUserId), обновление данных (UpdateUser) и удаление пользователя (DeleteUser).

Для передачи данных между клиентом и сервером используются DTO. Доступ к маршруту (api/User), а также операциям DeleteUser и GetUserId разрешён только пользователям с правами администратора. Работа модуля представлена на рисунке 8.

Рисунок 8 – Конечные точки User

Клиент будет использовать программу WPF (Windows Presentation Foundation) – это фреймворк для создания, современных десктопных приложений для Windows. Его работа будет заключатся в следующем: UI – главное в этом приложении, если будет удобен дизайн, то и будет проще делать заказы, а значит, больше прибыли, это все показывает как важно делать информативную и понятную разметку xaml для WPF, создание заказов в клиенте и их отправка на сервер по конечной точке API, а также принятие ответов от сервера. Его суть заключается быть почтальоном среди сервера и клиента. Для безопасности я буду использовать JWT токен и BCrypt для шифрования пароля и вот почему: пароли очень важный атрибут объекта пользователя, если БД взломают или сделают SQL инъекцию, то узнают пароли и испортят данные.

Сравнение выбора средств: BCrypt.Net-Next – безопасное хранение паролей C# (и .NET), это библиотека используется только для паролей она не шифрует отдельные данные, она медленная по сравнению с аналогами, но проста в использовании для системы, использует криптографическую хеш-функцию.

JWT – это безопасность, не нужно будет много раз входить в приложение если что-то случится например перебои с интернетом, чтобы повар и официант не смогли менять цены и позиции меню, в токен вшита роль входящего сотрудника. В сравнении с cookie выбранный JWT имеет несколько преимуществ: сервер не

хранит состояние сессии что снижает загрузку, проверка токена происходит на сервере, не надо будет обращаться к бд очень часто. Токен использует криптографическую подпись.

.NET 9 и asp.net core, потому что они идеально подходят к созданию API, фреймворк очень удобен тем, что выполняет много автоматических задач, например он маршрутизирует запросы, какой контроллер обрабатывает HTTP запрос (GET, POST, PUT, DELETE). Преобразует объекты в форматы JSON для отправки клиенту и обратно десериализует. Управление жизненным циклом с помощью middleware – конвейером. И предоставляет готовые ресурсы для создания точек.

MS SQL – реляционная БД, хорошо работает с .NET, несложная, можно управлять через интерфейс в программе SQL Server, через визуал проще взаимодействовать с данными и управлять ими, так же возможно подключить к Visual Studio и управлять ее данными через IDE. Паттерн я выбрал чтобы каждый слой не знал друг о друге, система немаленькая и так проще будет тестировать каждый блок и чинить в случае поломки.

Аналоги: Windows Presentation Foundation указаны в списке ниже.

1. WPF – мощный фреймворк для десктопных приложений, работает с .NET на чем и работает автоматизированная система, оптимизирована под API. По сравнению с аналогами он гарантированно хороший выбор.

2. UWP - Universal Windows Platform довольно устаревшая система, на замену ему пришел WinUI 3, он менее гибкий по сравнению с WPF или WinUI, также не совместим с меньше 10 версией Windows.

3. Avalonia UI – мощный фреймворк для создания десктопных приложений на windows похож на WPF но он является сторонним решением (не от Microsoft) и ориентирован в первую очередь на кроссплатформенность (Linux, macOS).

Таким образом, выбранный стек технологий (ASP.NET Core Web API + MS SQL Server + WPF) полностью соответствует техническим требованиям проекта, обеспечивает необходимый уровень безопасности (JWT) и позволяет реализовать автоматизированную систему для предприятия общественного питания.

Макеты приложения:

- страница входа в приложение показана на рисунке 9, где представлен стандартный вход в приложение с логином и паролем;
- страница меню указана на рисунке 10, здесь мы видим меню, заказы, категории;
- страница администратора указан на рисунке 11. Он управляет профилями пользователей;
- страница заказа указана на рисунке 12, редактирование, удаление, добавление заказа, полное управление, деление на разные счета.

Все страницы будут реализованы в приложении WPF автоматизированной системы. Представленный интерфейс является примерным дизайном каждого экрана приложения. Будет обеспечено синхронное отображение данных, поступающих с серверной части системы. Приложение будет использоваться как основной инструмент для управления автоматизированной системой кафе.

1. На рисунке 9 изображён макет страницы входа в систему, предназначенной для авторизации пользователей путём ввода логина и пароля.

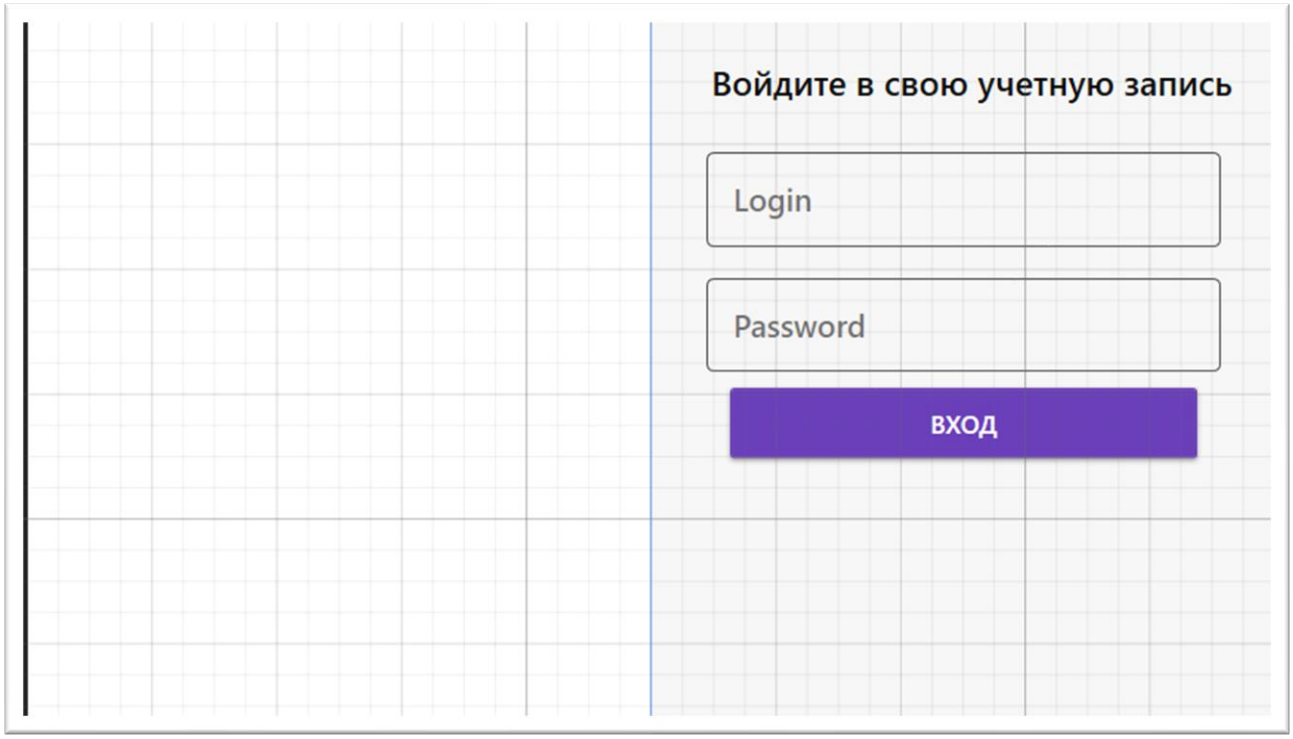


Рисунок 9 - Страница входа

2. На рисунке 10 представлен макет главного меню приложения, используемого официантом, поваром и администратором.

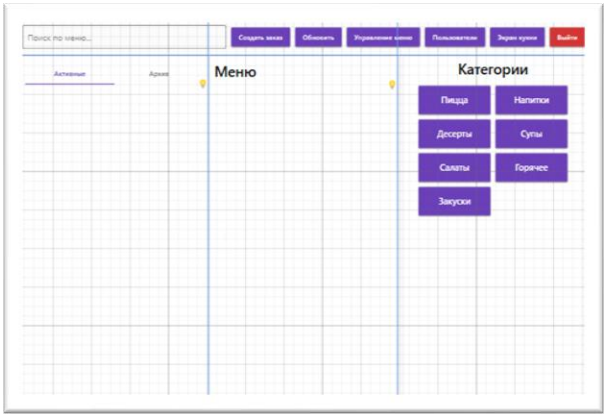


Рисунок 10 - Страница меню

3. На рисунке 11 изображён макет страницы управления пользователями, доступной только для администратора системы.

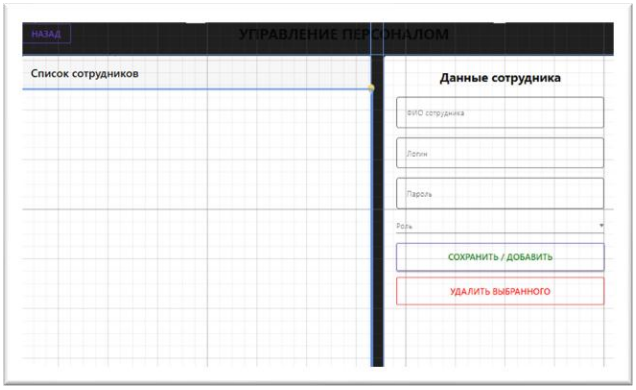


Рисунок 11 - Изменение пользователей

4. На рисунке 12 представлен макет страницы выбранного заказа, используемой официантом, поваром и администратором.

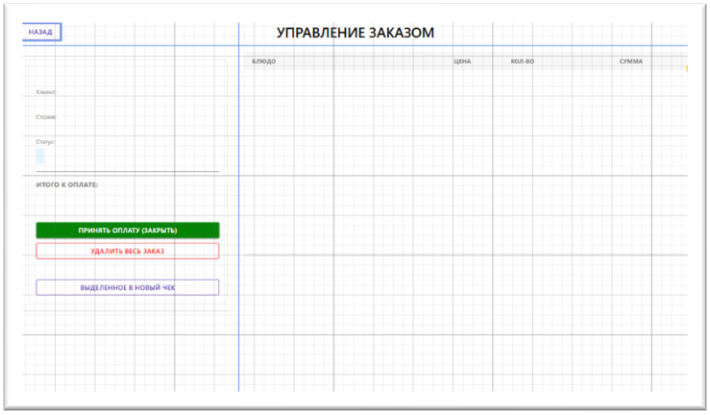


Рисунок 12 - Страница заказа

3 Разработка и тестирование

Разработка программного продукта велась с использованием многоуровневой архитектуры (N-Layer), что позволяет разделить ответственность между компонентами системы. Реализация серверной части (ASP.NET Core Web API) построена на основе трех слоев, а именно:

1. Web Layer (Controllers) – принимает HTTP-запросы.
2. Service Layer (Services) – содержит бизнес-логику.
3. Data Access Layer (Repositories) – взаимодействует с базой данных через Entity Framework Core.

Полный исходный код программного продукта размещён в репозитории проекта и приведён в приложении А.

На рисунке 13 показано то, как в файле ***Program.cs*** настроены внедрение зависимостей (DI - Dependency Injection) Что связывает интерфейсы и их реализацию.

```
builder.Services.AddScoped<IMenuItemRepository, MenuItemRepository>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IOrderRepository, OrderRepository>();
builder.Services.AddScoped<IOrderItemRepository, OrderItemRepository>();

builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IOrderService, OrderService>();
builder.Services.AddScoped<ITokenService, TokenService>();
```

Рисунок 13 - DI

Модуль аутентификации и безопасности реализован с использованием JWT-токенов. Сервис TokenService отвечает за генерацию токена, содержащего идентификатор пользователя, логин и роль. На рисунке 14 изображена реализация данного механизма.

```
var claims = new List<Claim>
{
    new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
    new Claim(ClaimTypes.Name, user.Login),
    new Claim(ClaimTypes.Role, user.RoleId.ToString())
};
```

Рисунок 14 - Данные пользователя в токене

Одной из основных функций системы является обработка заказов. Логика данного модуля реализована в сервисе ***OrderService***. Метод создания заказа включает проверку корректности выбранных блюд, расчёт общей стоимости и сохранение данных в базе данных. На рисунке 15 изображена реализация механизма обработки заказов.

```
public async Task<OrderResponseDto> CreateOrderAsync(CreateOrderDto orderDto)
{
    //Создаем новый заказ
    var order = new Order
    {
        UserId = orderDto.UserId,
        TableNumber = orderDto.TableNumber,
        Status = orderDto.Status,
        CreatedAt = DateTime.UtcNow
    };
    //Добавляем блюда в заказ
    foreach (var item in orderDto.Items)
    {
        var menuItem = await _menuItemRepository.GetMenuItemByIdAsync(item.MenuItemId);
        if(menuItem == null || menuItem.Available == false)
        {
            throw new Exception($"Блюдо {item.MenuItemId} не найдено. или же в стоп листе");
        }
        var newOrderItem = new OrderItem
        {
            UnitPrice = menuItem.Price,
            MenuItemId = item.MenuItemId,
            Quantity = item.Quantity,
            MenuItem = menuItem
        };
        order.OrderItems.Add(newOrderItem);
    }
    await _orderRepository.CreateOrderAsync(order);
    // Формируем ответ
    var orderResponse = new OrderResponseDto
    {
        OrderId = order.OrderId,
        UserId = order.UserId,
        TableNumber = order.TableNumber,
        CreatedAt = order.CreatedAt,
        Status = order.Status,
        TotalAmount = order.OrderItems.Sum(oi => oi.UnitPrice * oi.Quantity),

        Items = order.OrderItems.Select(oi => new OrderItemDto
        {
            OrderItemId = oi.OrderItemId,
            OrderId = oi.OrderId,
            MenuItemId = oi.MenuItemId,
            Quantity = oi.Quantity,
            UnitPrice = oi.UnitPrice,
            MenuItemName = oi.MenuItem?.Name ?? "Неизвестно"

        }).ToList()
    };
    return orderResponse;
}
```

Рисунок 15 - Создание заказа

Серверная часть приложения развертывается в Docker-контейнере. Для обеспечения переносимости, изоляции, упрощения процесса использовалась контейнеризация Docker. Это гарантирует одинаковое поведение приложения как на локальной машине разработчика, так и на сервере. Конфигурация контейнера описана в Dockerfile, представленном на рисунке 16.

```
# Образ для запуска сервера
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

# Образ для сборки приложения
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["CafeAPI/CafeAPI.csproj", "CafeAPI/"]
RUN dotnet restore "./CafeAPI/CafeAPI.csproj"
COPY . .
WORKDIR "/src/CafeAPI"
RUN dotnet build "./CafeAPI.csproj" -c $BUILD_CONFIGURATION -o /app/build

# Сборка и публикация приложения
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./CafeAPI.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

# Финальный этап – создание исполняемого образа
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "CafeAPI.dll"]
```

Рисунок 16 – Dockerfile

В среде Docker показана работа двух контейнеров, объединённых в одной сборке docker-compose: база данных **cafedb** и серверная часть **cafeapi**. На рисунке 17 представлена конфигурация данной среды.

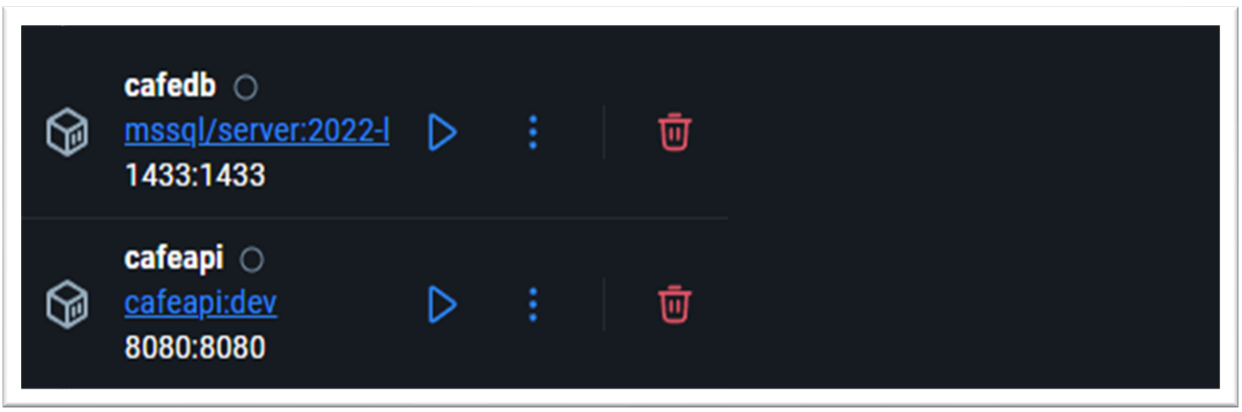
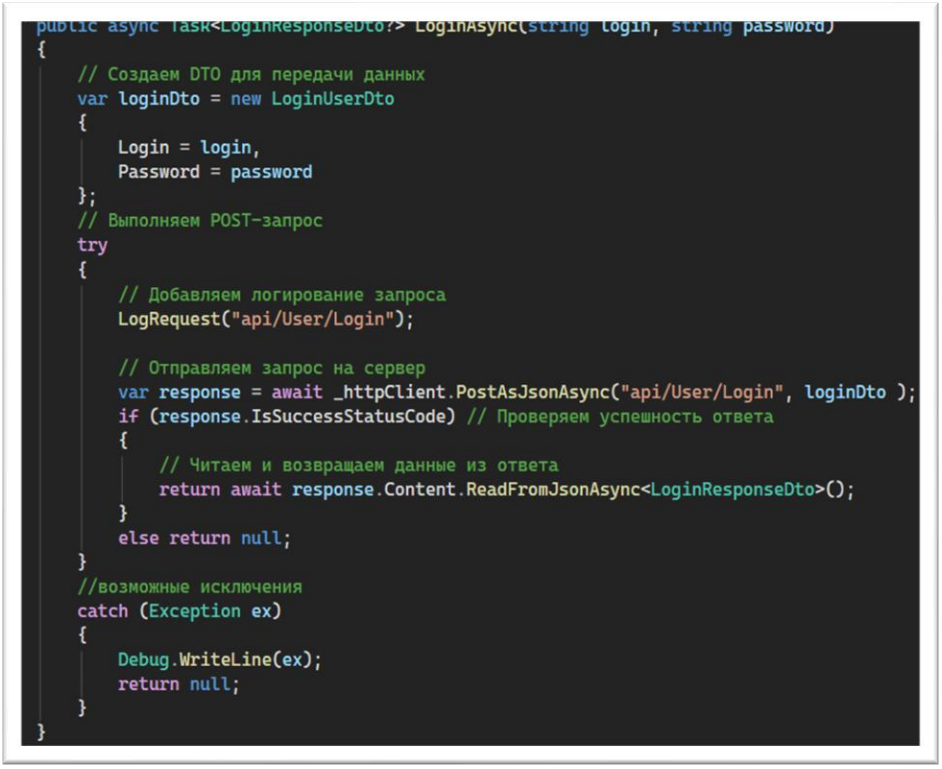


Рисунок 17 – Сборка контейнеров

В клиентском приложении передача данных реализована с помощью класса **ApiService**, который взаимодействует с сервером через экземпляр **_httpClient** для управления данными. На рисунке 18 представлен пример метода авторизации.

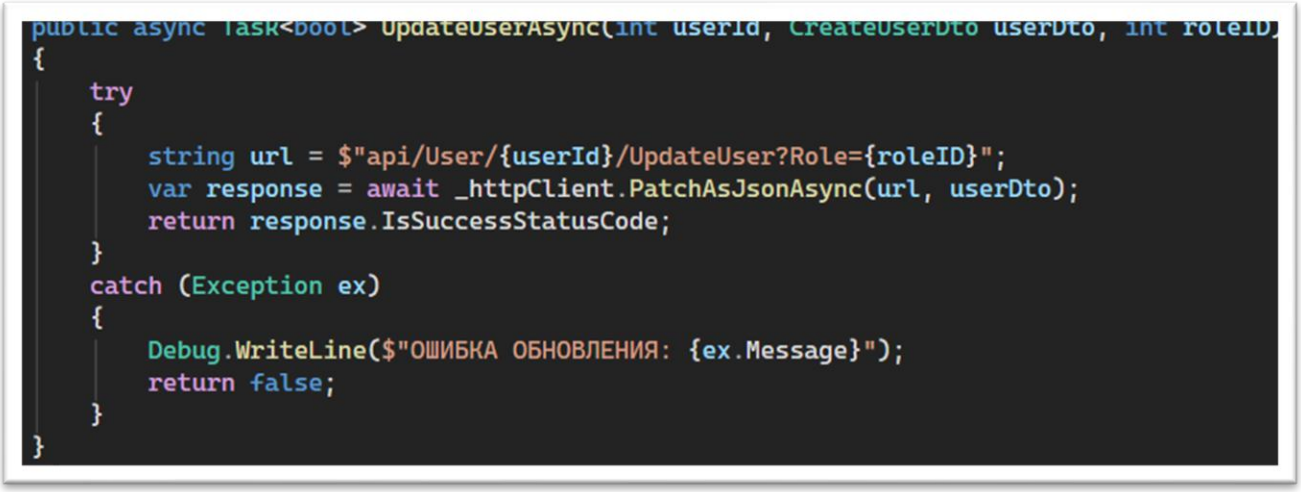


```
public async Task<LoginResponseDto?> LoginAsync(string login, string password)
{
    // Создаем DTO для передачи данных
    var loginDto = new LoginUserDto
    {
        Login = login,
        Password = password
    };
    // Выполняем POST-запрос
    try
    {
        // Добавляем логирование запроса
        LogRequest("api/User/Login");

        // Отправляем запрос на сервер
        var response = await _httpClient.PostAsJsonAsync("api/User/Login", loginDto );
        if (response.IsSuccessStatusCode) // Проверяем успешность ответа
        {
            // Читаем и возвращаем данные из ответа
            return await response.Content.ReadFromJsonAsync<LoginResponseDto>();
        }
        else return null;
    }
    //возможные исключения
    catch (Exception ex)
    {
        Debug.WriteLine(ex);
        return null;
    }
}
```

Рисунок 18 - Авторизация

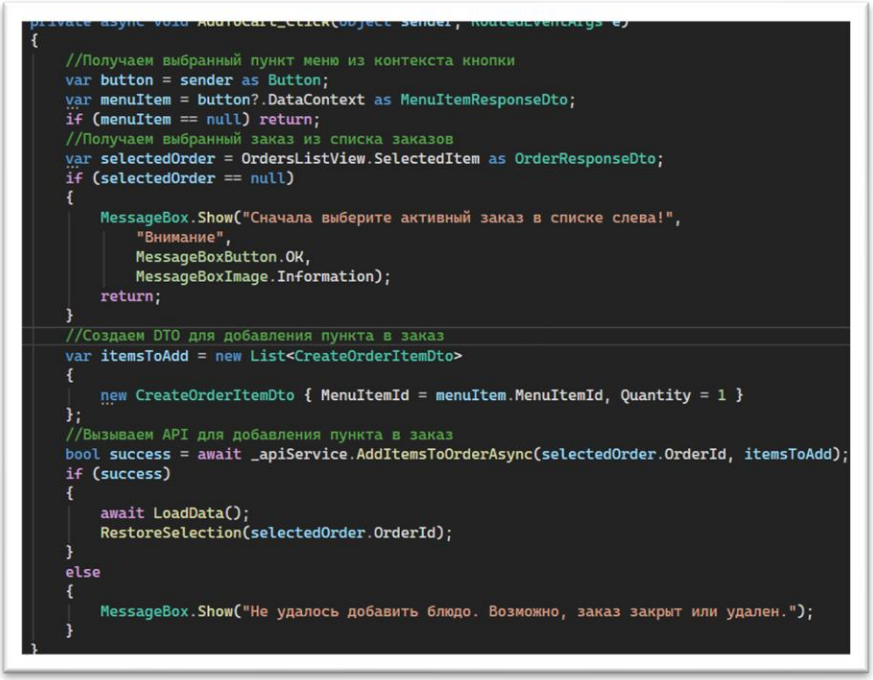
Метод обновления профиля реализован так, что приложение берет ссылку на соответствующий эндпоинт, сериализует объект с данными пользователя и отправляет его на сервер. Демонстрация работы метода представлена на рисунке 19.



```
public async Task<bool> UpdateUserAsync(int userId, CreateUserDto userDto, int roleID)
{
    try
    {
        string url = $"api/User/{userId}/UpdateUser?Role={roleID}";
        var response = await _httpClient.PatchAsJsonAsync(url, userDto);
        return response.IsSuccessStatusCode;
    }
    catch (Exception ex)
    {
        Debug.WriteLine($"ОШИБКА ОБНОВЛЕНИЯ: {ex.Message}");
        return false;
    }
}
```

Рисунок 19 - Обновление профиля

Помимо функций авторизации и обновления профиля, важной частью клиентского приложения является добавление позиций в заказ на странице MainPage. В файле **MainPage.xaml.cs** реализована логика добавления выбранных позиций меню в заказ. Демонстрация работы данной функции приведена на рисунке 20.



```
{
    //Получаем выбранный пункт меню из контекста кнопки
    var button = sender as Button;
    var menuItem = button?.DataContext as MenuItemResponseDto;
    if (menuItem == null) return;
    //Получаем выбранный заказ из списка заказов
    var selectedOrder = OrdersListView.SelectedItem as OrderResponseDto;
    if (selectedOrder == null)
    {
        MessageBox.Show("Сначала выберите активный заказ в списке слева!",
            "Внимание",
            MessageBoxButton.OK,
            MessageBoxImage.Information);
        return;
    }
    //Создаем DTO для добавления пункта в заказ
    var itemsToAdd = new List<CreateOrderItemDto>
    {
        new CreateOrderItemDto { MenuItemId = menuItem.MenuItemId, Quantity = 1 }
    };
    //Вызываем API для добавления пункта в заказ
    bool success = await _apiService.AddItemsToOrderAsync(selectedOrder.OrderId, itemsToAdd);
    if (success)
    {
        await LoadData();
        RestoreSelection(selectedOrder.OrderId);
    }
    else
    {
        MessageBox.Show("Не удалось добавить блюдо. Возможно, заказ закрыт или удален.");
    }
}
```

Рисунок 20 – Добавления товара

Тестирование программного продукта проводилось с целью проверки корректности работы реализованных модулей, устойчивости системы к ошибочным действиям пользователя, а также соответствия функциональных возможностей заявленным требованиям.

Методика тестирования осуществлялась по принципу *черного ящика*, при котором проверялась работа системы без анализа внутренней реализации кода. Тестирование выполнялось параллельно с процессом разработки и включало ручную проверку серверной и клиентской частей приложения. Для комплексной оценки работоспособности были разработаны тестовые сценарии, охватывающие основные функции автоматизированной системы. В ходе тестирования были реализованы и проверены механизмы валидации пользовательских действий.

На рисунке 21 изображён механизм проверки перед добавлением позиции меню в заказ. Приложение сначала проверяет, выбран ли текущий заказ: если идентификатор заказа отсутствует, действие блокируется, а пользователь получает уведомление. Аналогичные проверки реализованы на сервере в контроллерах, что предотвращает отправку неполных или некорректных данных.

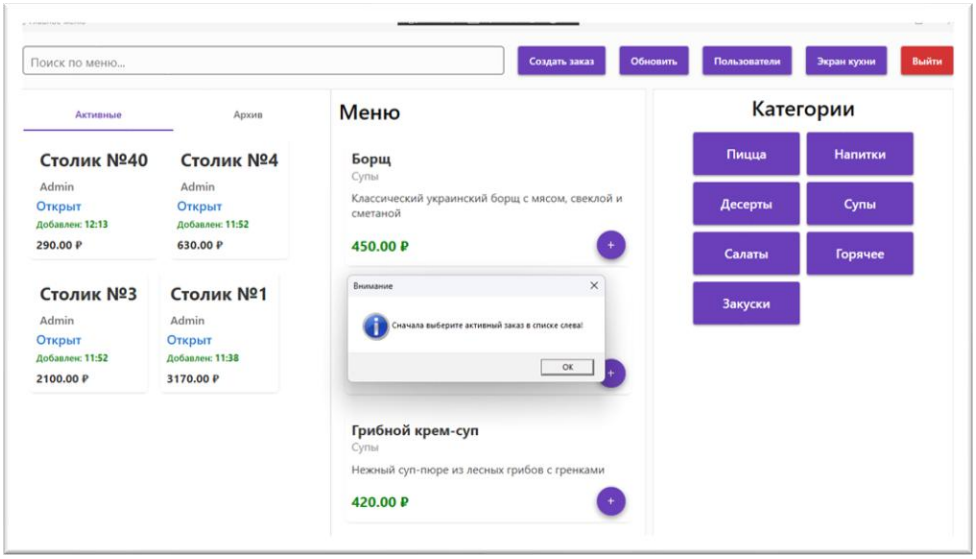


Рисунок 21 – Проверка добавления заказа

На рисунке 22 показан механизм проверки занятости стола. Приложение не позволяет закрепить за один стол два заказа, обеспечивая корректность распределения заказов по столам.

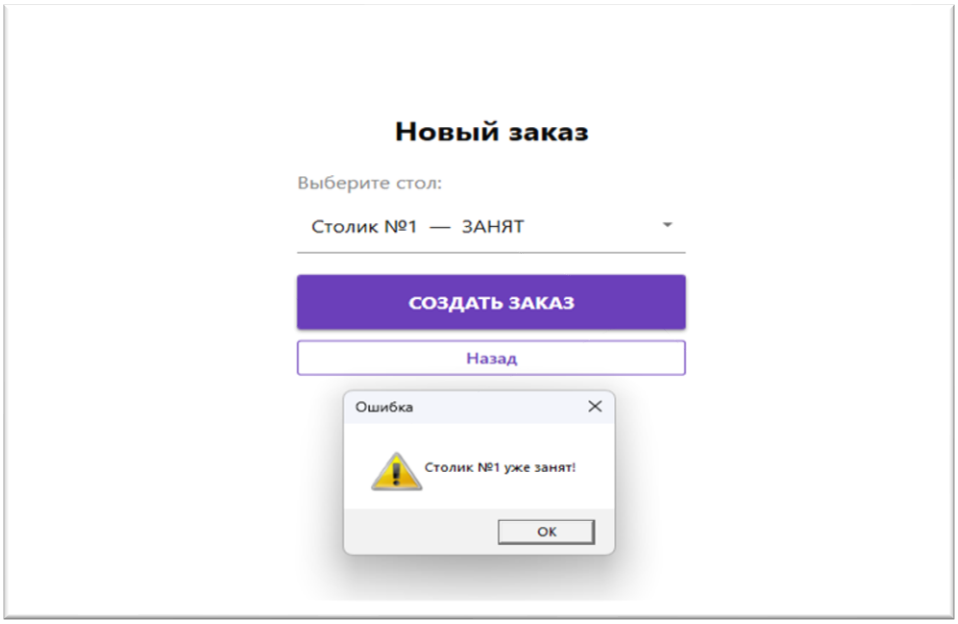


Рисунок 22 – Проверка стола.

На рисунке 23 показан механизм удаления блюд из архива, а именно, что кнопки для удаления спрятаны. Данная проверка важна для корректного расчёта конечной выручки, так как обеспечивает точное суммирование всех заказов без учёта удалённых позиций.

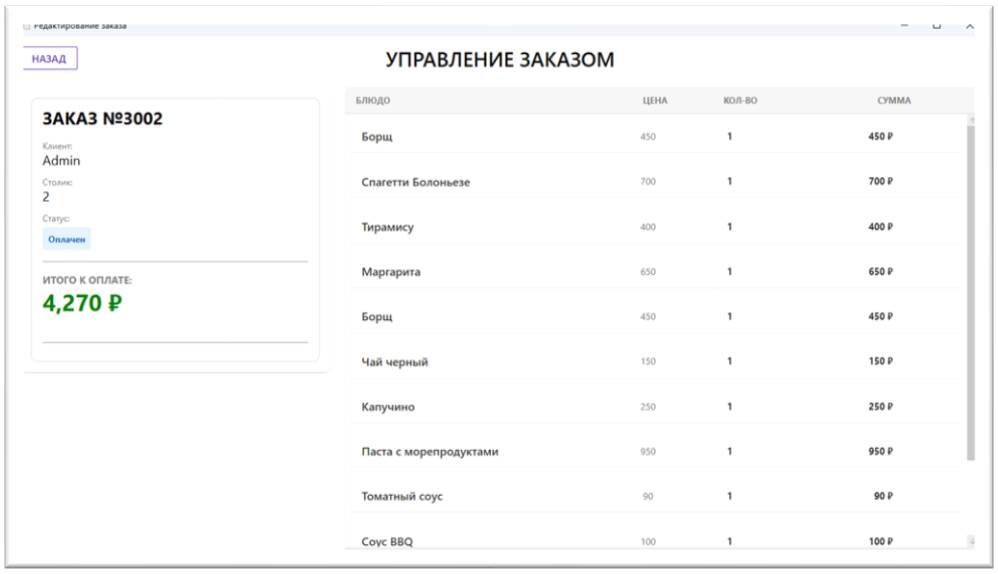


Рисунок 23 - Удаление из архива

На рисунке 24 показан механизм проверки статуса заказа. Если заказ успешно выполнен или оплачен, он перемещается в архив и перестаёт отображаться на странице кухни, что обеспечивает актуальность информации для персонала.

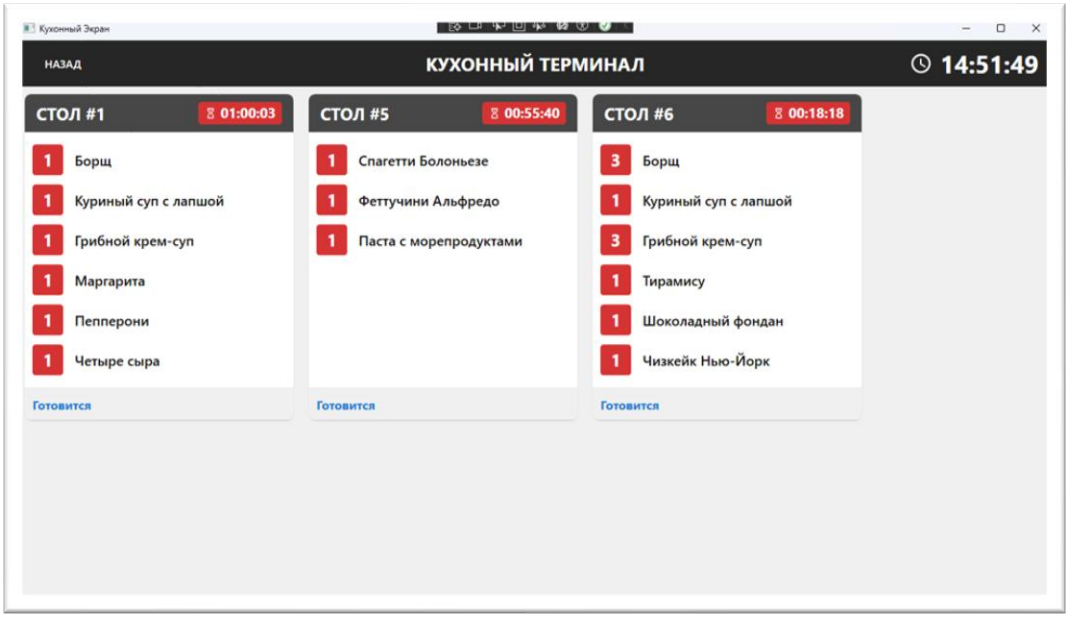


Рисунок 24 - Статус заказа кухня

Тестирование API-сервера проводилось вручную. Каждый эндпоинт проверялся путём отправки HTTP-запросов с различными входными данными и анализа ответов сервера.

Для этого использовался инструмент *Swagger*, который автоматически формирует описание доступных эндпоинтов и подставляет шаблоны JSON-запросов для их заполнения. Это упрощает процесс тестирования, так как позволяет быстро отправлять запросы и сразу получать коды ответов сервера.

Клиентское приложение на WPF и тестировщик анализируют HTTP-коды ответов и соответствующим образом реагируют на результат выполнения запроса. Основные коды ответов сервера приведены в таблице 2.

Таблица 2 - Таблица ответов сервера с расшифровкой кодов.

Код	*Название*	*Описание*
200	OK	Запрос успешно выполнен
201	Created	Ресурс успешно создан
400	Bad Request	Ошибка валидации входных данных
401	Unauthorized	Отсутствует или некорректен токен авторизации
403	Forbidden	Недостаточно прав для выполнения операции
404	Not Found	Запрошенный ресурс не найден
500	Internal Server Error	Внутренняя ошибка сервера

Таким образом, использование стандартных HTTP-кодов позволяет упростить обработку результатов запросов как на стороне сервера, так и на стороне клиентского приложения. Это упрощает реализацию логики обработки ошибок, повышает читаемость кода и делает систему более устойчивой к некорректным входным данным и ошибкам выполнения.

Заключение

В ходе выполнения курсового проекта была выполнена цель. Создание автоматизированной системы, для оптимизации процессов обслуживания клиентов в кафе.

Проведен анализ предметной области. Была спроектирована архитектура будущей системы. Выбрана клиент-серверная архитектура с использованием REST API, обеспечивающая гибкость и масштабируемость решения.

Реализована база данных и логическая структура. Спроектирована и создана база данных на MS SQL Server, включающая таблицы для хранения данных о пользователях, ролях, категориях меню и заказах. Взаимодействие приложения с базой данных организовано посредством ORM Entity Framework Core.

Реализована серверная часть приложения. Разработан API на платформе ASP.NET Core Web API. Внедрена система аутентификации и авторизации на основе JWT-токенов, обеспечивающая разграничение прав доступа. Созданы контроллеры для управления меню, обработки заказов и администрирования пользователей.

Разработаны клиентская часть и пользовательский интерфейс. Создано приложение на технологии WPF с понятным графическим интерфейсом. Реализована бизнес-логика, включая проверку занятости столиков в реальном времени для предотвращения дублирования заказов.

Дополнительно была обеспечена переносимость системы: для упрощения развертывания серверной части применена контейнеризация Docker.

Таким образом, в результате проделанной работы цель и все поставленные задачи были выполнены в полном объеме.

В дальнейшем разработанная автоматизированная система может быть расширена и доработана. Возможными направлениями развития являются внедрение системы на POS-терминалы и, также улучшить архитектуру клиентского приложение добавление паттерна MVVM.

Список использованной литературы

1. Макдональд М. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов [Текст] / М. Макдональд. – Москва: Вильямс, 2013. – 1024 с.
2. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения [Текст] / Р. Мартин. – Санкт-Петербург: Питер, 2021. – 352 с.
3. Прайс М. C# 9 и .NET 5. Разработка и оптимизация [Текст] / М. Прайс. - Санкт-Петербург: Питер, 2022. – 848 с.
4. Троелсен Э. Язык программирования C# 9 и платформа .NET 5 [Текст] / Э. Троелсен, Ф. Джепикс. – Москва: Вильямс, 2022. – 1328 с.
5. Фримен А. ASP.NET Core для профессионалов [Текст] / А. Фримен. – Москва: Вильямс, 2019. – 912 с.
6. Документация по службе «Управление API» [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/azure/api-management/ (дата обращения 10.10.2025).
7. Документация по ASP.NET Core [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/core/ (дата обращения 10.11.2025).
8. Документация по Entity Framework Core [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/ef/core/ (дата обращения 11.11.2025).
9. Руководство по языку C# [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения 13.01.2026).

Приложения

Приложение А. Исходный код

Репозиторий проекта:

https://github.com/Bikmacs/CafeSystem.

На рисунке 25 QRCode репозитория

Рисунок 25 - QRCode

Приложение Б. UI сервера

Для документирования и тестирования серверной части (Backend) используется инструмент Swagger UI. Он предоставляет графический интерфейс для отправки HTTP-запросов к API и просмотра схем данных.

Полная спецификация разработана в формате OpenAPI 3.0 и опубликована с использованием инструментов Swagger UI на платформе GitHubPages. Это позволяет разработчикам фронтенда и тестировщикам взаимодействовать с сервером удаленно, без необходимости разворачивать локальное окружение. Ссылка: https://bikmacs.github.io/cafeSwagger/, QR-Code ведущий на сайт с описанием эндпоинтов. На рисунке 26.

Рисунок 26 - QRCode

Приложение В. Интерфейс программы

На рисунке 27 представлена карта приложения, демонстрирующая навигацию по страницам клиентского приложения.

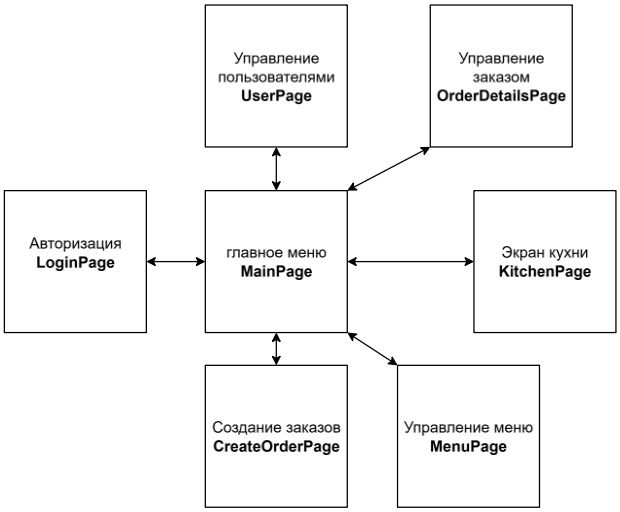


Рисунок 27 - Карта приложения

На рисунке 28 показана схема взаимодействия элементов приложения, иллюстрирующая связи, последовательность действий и поток данных между компонентами программы.

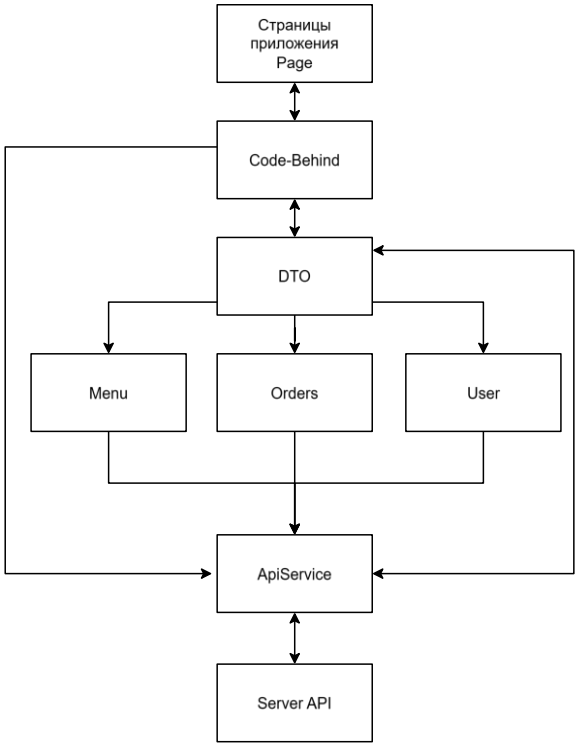


Рисунок 28 – Схема приложения

Приложение Г. База данных

Диаграмма базы данных кафе с типом данных и null значением полей, и указанием связей, продемонстрирована на рисунке 29.

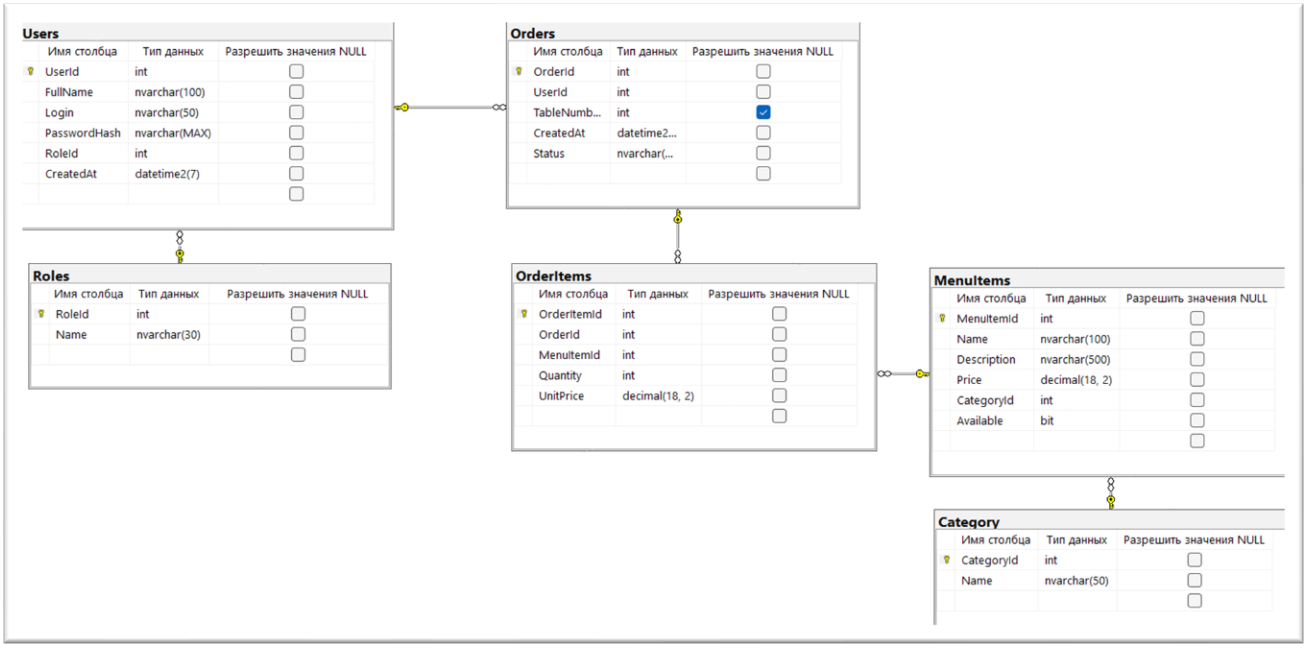


Рисунок 29 - Диаграмма баз данных

Зарезервированные данные, используемые для автоматического заполнения базы данных при инициализации, определены в контексте базы данных сервера в классе *CafeDbContext* и доступны по ссылке: https://github.com/Bikmacs/CafeSystem/blob/main/CafeSolution/CafeAPI/Data/CafeDbContext.cs. QR Code. Рисунок 30.

Рисунок 30 - Qrcode