МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

**РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ ПИЦЦЕРИИ**

**Пояснительная записка**

Отчёт по дисциплине

«Объектно-ориентированное программирование»

ТПЖА.090302.009 ПЗ

Разработал студент гр. ИТб-2301-01-00 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Зуев А. И./

(подпись)

Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Кузьминых М.М./

(подпись)

Работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_» \_\_\_\_\_\_\_\_\_\_2022 г.

Киров 2022



**Реферат**

Зуев А.И. Реализация информационной системы пиццерии: ТПЖА.090302.009 ПЗ: Отчет по курсовой работе / ВятГУ, каф. САУ; рук. М.М. Кузьминых – Киров, 2022. ПЗ 49 с., 29 рис., 1 табл., 5 прил.

ПЕРЕЧЕНЬ КЛЮЧЕВЫХ СЛОВ: ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ИНФОРМАЦИОННАЯ СИСТЕМА, C#, ОБЪЕКТНО-ОРИЕНТИРОВННОЕ ПРОГРАММИРОВАНИЕ, ENTITY FRAMEWORK, БАЗА ДАННЫХ.

Объект исследования – информационная система пиццерии.

Предмет исследования – объекты и процессы информационной системы.

Цель работы – автоматизация и построение информационной системы предметной области с использованием подхода ООП и языка программирования C#.

В ход выполнения работы были использованы: C#, PostgreSQL, Entity Framework, Visual Studio.

В разработанном web-приложении авторизованный сотрудник имеет возможность или вести учёт, редактировать и добавлять данные о пиццах, закусках, сотрудниках, или добавлять в заказ блюда, выбранные клиентом, - в зависимости от типа сотрудника.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Содержание**  Изм.  Лист  № докум.  Подпись  Дата  Лист  2  ТПЖА.090302.009 ПЗ  Разраб.  *Зуев А.И.*  Провер.  Кузьминых М.М.  Реценз.  Н. Контр.  Утверд.  *Реализация информационной системы пиццерии*  Литер  Листов  32  *Кафедра САУ,*  *группа ИТб-2301-01-00* | | |  | |
| Введение………………………………………………………………..……… | | | 3 | |
| 1. Анализ предметной области…………………………………………..…... | | | 4 | |
| 1.1 Процессы протекающие в предметной области…….……………..…. | | | 4 | |
| 1.2 Классы предметной области....………….………………………..…..... | | | 4 | |
| 2. Реализация классов на языке программирования C#........... ……...……... | | | 6 | |
| 3. Создание базы данных с использованием Entity Framework…..………... | | | 7 | |
| 3.1 Настройка обращения к базе данных …………………..……………..  3.2 Генерация сущностей базы данных…………………………………. | | | 7  7 | |
|  | |
|  | |
| 4. Разработка приложения, реализующего базовую функциональность..… | | | 9 | |
| 4.1 Выбор платформы для разработки …………………..……………..  4.2 Демонстрация приложения……………………………………………. | | | 9  9 | |
| Заключение……………………………………………………………….…... | | | 23 | |
| Приложение А (обязательное) Листинг реализации ключевых классов….. | | | 24 | |
| Приложение Б (обязательное) Листинг программного кода ApplicationContext……………………………………………………………. | | | 29 | |
| Приложение В (обязательное) Листинг миграций, сгенерированных Entity Framework……………………………………………….……………. | | | 30 | |
| Приложение Г (обязательное) Ключевые фрагменты кода, отвечающие за функционал программы……………………………………………….……. | | | 36 | |
| Приложение Д (справочное) Библиографический список…...…………….. | | | 49 | |
|  | | |
|  | | |  | |

# Введение

Движущей силой развития бизнеса в современных условиях становятся информационные технологии. Сегодня почти ни одно крупное предприятие не может обойтись без своего IT-сектора.

Автоматизация бизнес-проекта помогает избежать потерь времени и сократить издержки на решение различных задач, структурировать большие объёмы информации, а также предоставит обширные возможности эффективного управления бизнесом.

Разработка собственной информационной системы в краткосрочной перспективе является не лучшим вложением. Однако для серьёзного бизнеса, ориентированного на долгосрочное функционирование - вложение куда более выгодное. Воплотить такой замысел в жизнь можно только с собственной системой, которая будет учитывать всю специфику и детали нашего бизнеса, поскольку специализированное решение будет гораздо лучше универсального.

Доход со своей пиццерии – мечта многих предпринимателей. Этот бизнес считается довольно прибыльным, невзирая на высокую конкуренцию в этой области. Целевая аудитория заведения охватывает людей самых разных возрастных категорий и вкусовых предпочтений. Добиться максимальной производительности труда и, соответственно, прибыли при сохранении стабильного качества продукции можно только с помощью автоматизации и цифровизации производства.

# 

# Анализ предметной области

Пиццерия – это предприятие, предоставляющее широкий ассортимент готовых блюд. Пицца в таком заведении остаётся главным блюдом, но вместе с тем предлагаются популярные салаты, закуски и напитки.

Информационная система пиццерии предполагает хранение больших объёмов информации о компонентах меню, о заказах клиентов, поэтому бизнес-процессы пиццерии нуждаются в автоматизации.

Ключевым блюдом пиццерии однозначно является пицца. Клиент должен иметь возможность не только выбирать размер пиццы, но и добавлять дополнительные ингредиенты по своему выбору. Ассортимент не должен сводиться только к пиццам: напитки, десерты, салаты и горячие закуски так же должны быть предложены клиенту на усмотрение.

1.1 Процессы, протекающие в предметной области

Разработанная информационная система должна предназначаться для учёта деятельности различными типами сотрудников.

Администратор должен иметь возможность просматривать оформленные заказы и вносить изменения в меню пиццерии: добавлять новые блюда, ингредиенты к пицце и сами пиццы с возможностью отображения необходимых параметров. Помимо всего прочего, администратору пиццерии будет открыт доступ к редактированию данных рабочих пиццерии и добавлению новых.

Посетителю пиццерии предлагается актуальное меню. Как только клиент определится с выбором, то к нему должен подойти сотрудник пиццерии. Перед оформлением заказа он должен найти клиента в базе данных по номеру телефона, а если клиента ещё нет, то зарегистрировать, указав его имя. Авторизованный официант составляет заказ и, получив окончательно подтверждение, оформляет его. Больше не нужно выполнять расчёты вручную или на переносном калькуляторе, теперь автоматизированная система посчитает итоговую массу и сумму заказа, исходя из выбранных клиентом блюд, и выведет их на экран.

Автоматизация вышеперечисленного функционала позволит сэкономить финансы компании, аппаратные ресурсы и время сотрудников.

1.2 Классы предметной области

Выявленные классы предметной области указаны в таблице 1.

Таблица 1 — классы предметной области

|  |  |  |
| --- | --- | --- |
| № п/п | Категория | Возможные классы |
| 1 | Физические или материальные объекты | Пицца, закуски, дополнительный ингредиент |
| 2 | Спецификации, описания объектов | Роль сотрудника, тип закуски, размер пиццы |
| 3 | Места | Пиццерия |
| 4 | Транзакции | Оформление заказа, добавление пиццы |
| 5 | Элементы транзакций |  |
| 6 | Роли людей | Администратор, официант |
| 7 | Контейнеры других объектов | Заказ клиента |
| 8 | Содержимое контейнеров | Пицца, закуски, дополнительный ингредиент |
| 10 | Организации | Пиццерия |
| 11 | События | Выбор компонентов меню, добавление компонентов меню |
| 12 | Процессы | Оформление заказа |

Используемые классы представлены в диаграмме на рисунке 1.

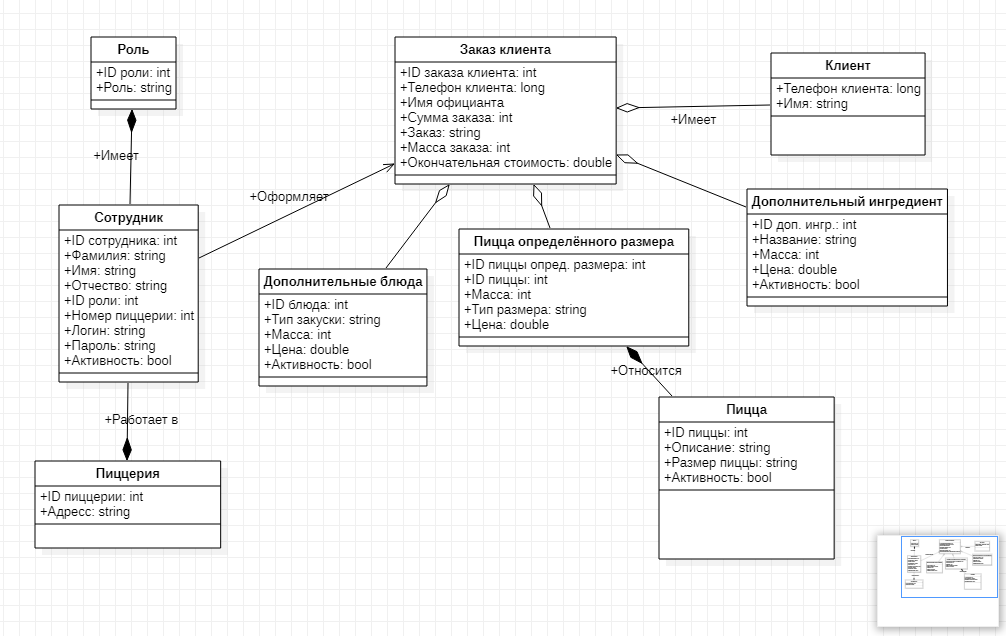


Рисунок 1 – Диаграмма классов UML

В данном разделе произведен анализ предметной области. Определены протекающие в ней процессы, подлежащие автоматизации, определены классы в соответствии с их категориями. Создана диаграмма UML с отображением классов и связи между ними.

# 2. Реализация классов на языке программирования C#

C Sharp – объектно-ориентированный язык программирования, имеющий в своём арсенале широкий набор библиотек. Из-за большого разнообразия синтаксических конструкций и возможности работать с платформой .NET, C# позволяет быстрее, чем любой другой язык, разрабатывать программные решения. Именно на этом языке будет написано приложение.

В предметной области фигурируют следующие классы: клиент, заказ клиента, пиццерия, сотрудник пиццерии, роль, пицца, пицца определённого размера, дополнительные блюда, дополнительные ингредиенты. В библиотеке классов реализованы следующие виды отношений: Ассоциация, Агрегация и Композиция.

Примером ассоциации является отношение между классами сотрудник и заказ клиента. Классы связаны, но слабо, время жизни экземпляров никак не зависит друг от друга.

Примером агрегации является отношение между классами заказ клиента и дополнительные блюда. Дополнительные блюда являются частью заказа клиента, но не обязательно содержится в нем, время жизни экземпляров не равно.

Примером композиции является отношение между классами роль и сотрудник. Сотрудник не может существовать без определённой ему роли в пиццерии, время жизни экземпляров равно.

В данном разделе рассмотрены предпосылки выбора языка для разработки приложения, ключевые классы предметной области и виды связей, встречающиеся между ними, а также приведены примеры их использования. Листинг программного кода классов представлен в приложении А.

**3. Создание базы данных с использованием Entity Framework**

В качестве библиотеки ORM используется Entity Framework. Для её изучения были использованы итернет-ресурсы Microsoft Docs и Metanit [1-2, приложение Г].

Entity Framework подходит для работы с базами данных в небольших приложениях. Для объектно-ориентированного подхода сопоставление объектов в программном обеспечении с таблицами в базе данных является крайне удобным, библиотека позволяет абстрагироваться от базы данных, реализуя отношения между классами.

Entity Framework поддерживает LINQ-запросы, отслеживание изменений и их сохранение в базе данных, а так же, что немаловажно, миграции схемы.

**3.1 Настройка обращения к базе данных**

В качестве СУДБ был выбран PostgreSQL – объектно-реляционная система управления базой данных, которая базируется на языке SQL. Он поддерживает многочисленные возможности, среди которых: надёжные механизмы транзакций, функции, триггеры, хранимые процедуры и многое другое. Для изучения тонкостей PostgreSQL была задействована документация компании Postgres Professional [3, приложение К] **.**

Для взаимодействия с базой данных был создан класс ApplicationContext, наследуемый от DbContext. В нем находится функционал для подключения к базе данных, а так же содержатся ссылки для обращения к сущностям базы данных. Листинг программного кода ApplicationContext приведен в приложении Б.

**3.2 Генерация сущностей в базе данных**

После создания моделей воспользуемся миграциями для генерации базы данных. Данный подход называется Code First – метод, предполагающий минимальное участие программиста в проектировании сущностей, большую часть работы выполняет Visual Studio и ORM. Код миграций, которые были сгенерированы Entity Framework представлен в приложении В. Физическая модель данных представлена на рисунке 2. На схеме отображены отношения, их атрибуты и связи.

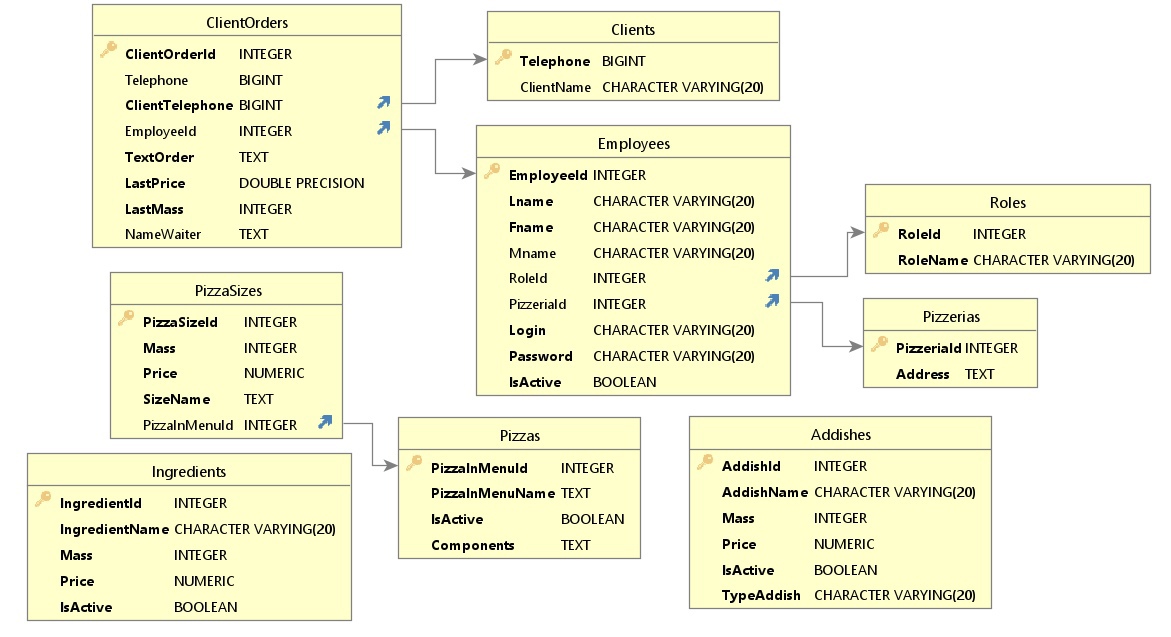


Рисунок 2 – Физическая модель данных

Таким образом, в этом разделе рассмотрены создание методов для взаимодействия с базой данных, создание самой базы данных и генерация сущностей с помощью миграций.

**4.** **Разработка приложения, реализующего базовую функциональность**

В данном разделе приведена демонстрация работы программы. Ключевые фрагменты кода, отвечающие за функционал программы приведены, в приложении Г.

**4.1 Выбор платформы для разработки**

Информационная система осуществляет работу с базой данных по средствам web-приложения. Пользователи системы работают с программой, используя обычный браузер. Доступность и высокая скорость интернета позволит работать с программным обеспечением, ничего не устанавливая его на локальный компьютер. А с помощью теговой разметки гораздо проще редактировать и обновлять программное обеспечение – достаточно внести правки на серверной стороне, и все сразу же смогут работать на обновлённой версии приложения.

В ходе разработки web-приложения не придётся создавать отдельные приложения для Mac OS, Windows, Linux или любой другой OC, не придётся продумывать решения под различные устройства – информационная система будет работать одинаково. Независимость от аппаратных решений и операционных систем, которую как раз таки обеспечивает работа в браузере, открывает новые возможности: например, работа через компьютерный планшет.

Active Server Page на базе технологии .NET от Microsoft (ASP.NET далее) – платформа для разработки web-приложений, поддерживающая работу с большинством распространённых СУБД: MS SQL Server, MySQL, Postgres. У языка C# есть официально встроенная поддержка для проектов ASP.NET. Для изучения основ работы с платформой для разработки была изучена документация Metanit [4, приложение К].

**4.2 Демонстрация приложения**

При запуске программы появляется окно авторизации (рисунок 3).

В зависимости от введённых данных – данных администратора или данных официанта, – открывается следующая страница. Если пользователь введёт некорректные данные, то будет подсвечено предполагаемое поле на той же странице, в котором он допустил ошибку (рисунок 4).

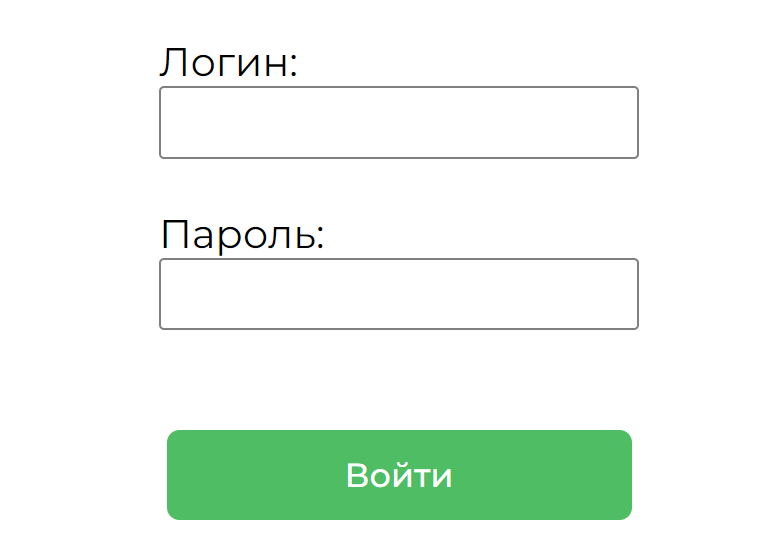


Рисунок 3 – Окно авторизации

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Окно авторизации с допущенной ошибкой

Если введённые в поле значения соответствуют хранящимся в базе данным, то пользователь увидит страницу приветствия. Первая страница для авторизованного администратора представлена на рисунке 5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Страница приветствия администратора

С помощью боковой панели, доступной отовсюду, администратор может перемещаться по страницам с различным функционалом.

Страница «Сотрудники» (рисунок 6) содержит таблицу с данными нанятых официантов и две кнопки: «Добавить» и «Восстановить».

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 6 – Страница с нанятыми сотрудниками

Кнопка «Добавить» пересылает на страницу с формой регистрации нового официанта в пиццерии (рисунок 7) и впоследствии предоставления ему доступа к приложению. Ни одна форма в приложении не допускает отправку пустых значений (не считая тех полей, где это предусмотрено: например, поле отчества). Браузер предупредит пользователя о неверно введённом значении (рисунок 8) сообщением под полем ввода, которое будет выделено цветом.

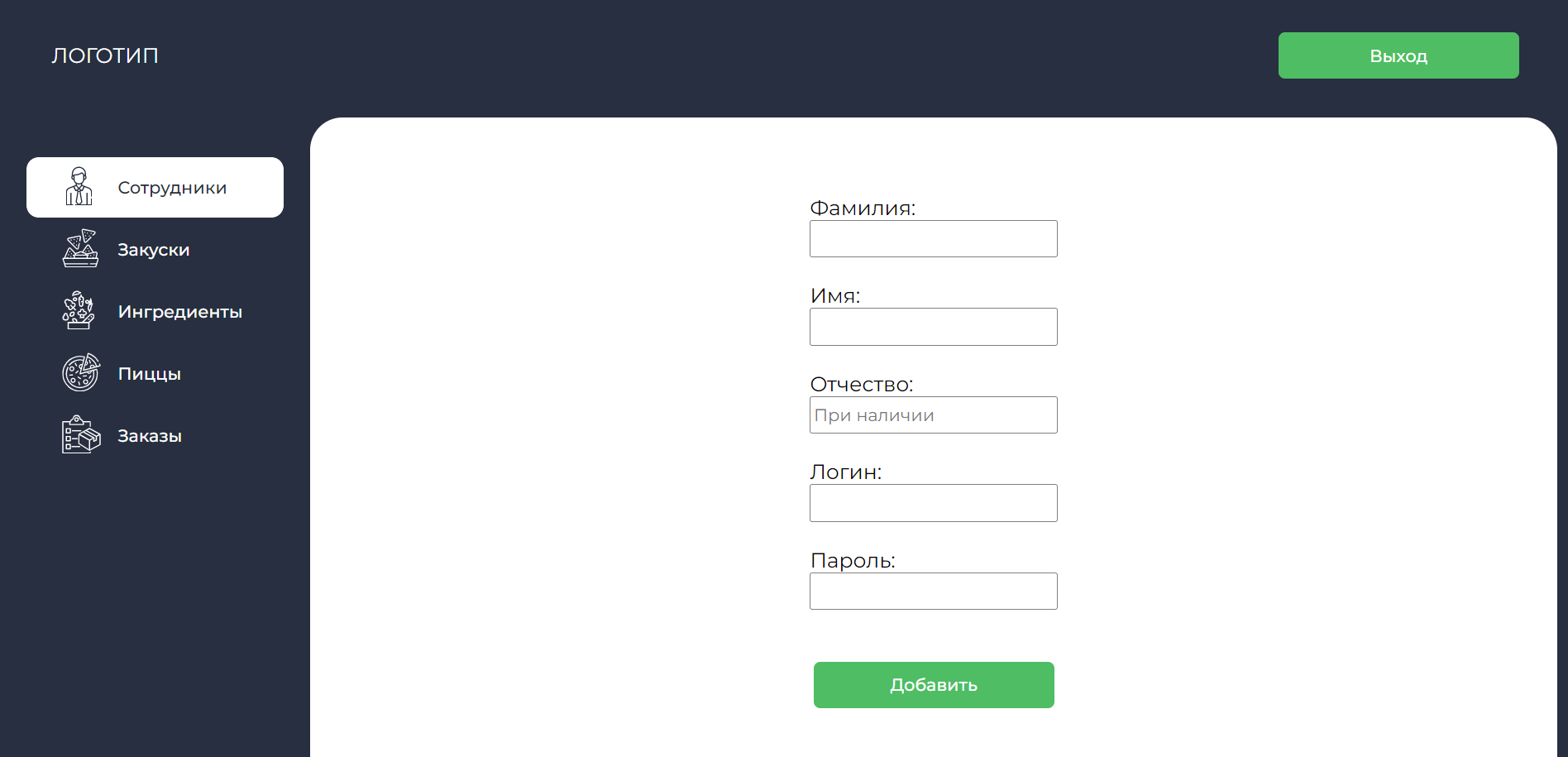
****

Рисунок 7 – Страница с формой регистрации нового официанта

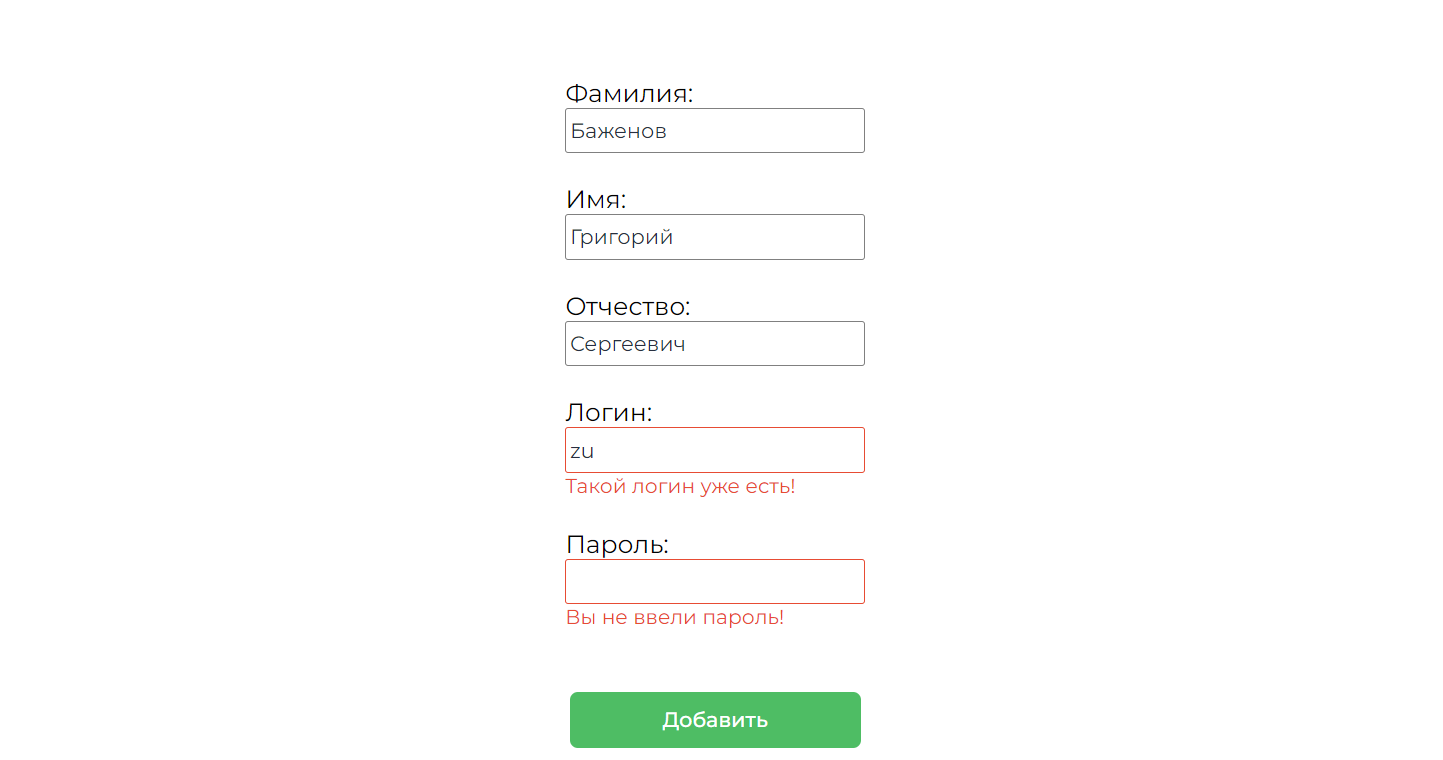
****

Рисунок 8 – Форма регистрации нового официанта с подсвеченными ошибками при вводе данных

Кнопка «Восстановить» перенаправляет на страницу с таблицей неработающих в пиццерии (рисунок 9), которым при входе будет отказано в доступе к приложению. Здесь можно увидеть данные сотрудника и легко восстановить его по клику на выделенную жирным шрифтом кнопку «Восстановить» в соответствующей строке таблицы. Тогда доступ к приложению будет восстановлен и официант снова сможет оформлять заказы клиентов.

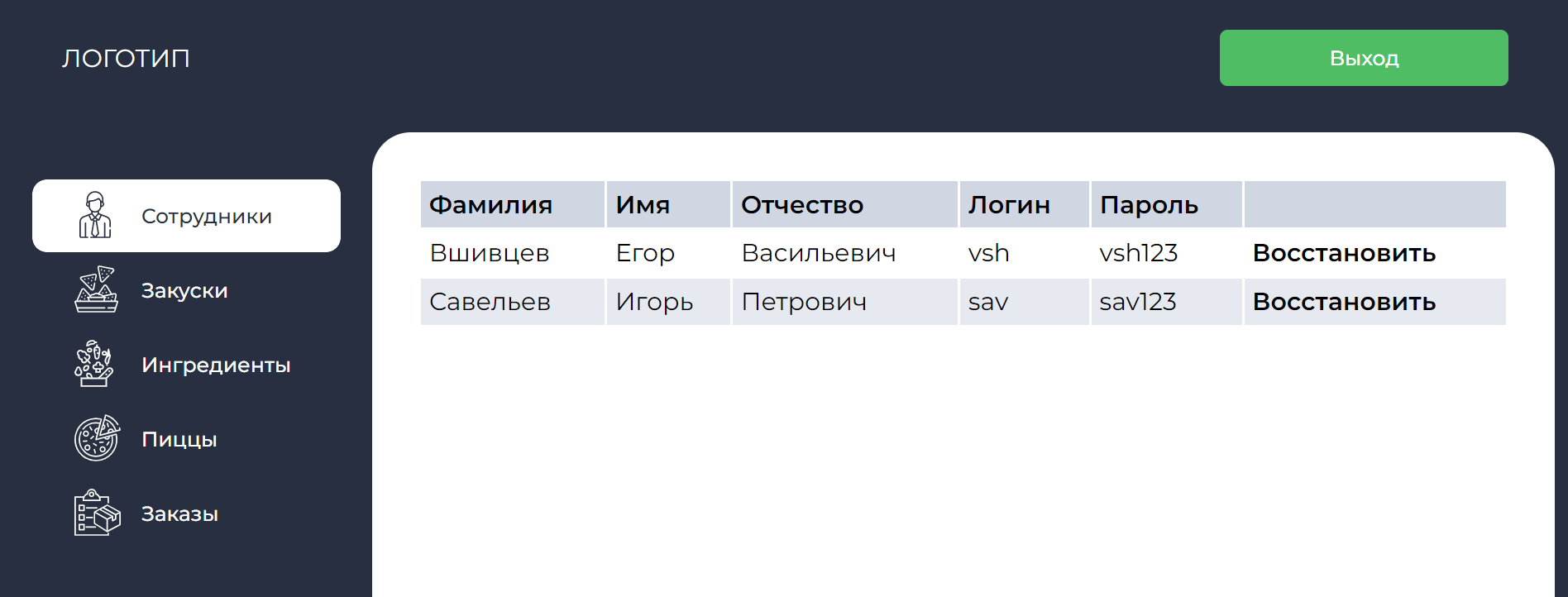


Рисунок 9 – Страница восстановления сотрудников пиццерии

В каждой строке таблицы данных рабочего на странице с нанятыми сотрудниками (рисунок 6) располагаются две выделенные жирным шрифтом кнопки: «Изменить» и «Уволить».

Кнопка «Изменить» перенаправит администратора на форму редактирования данных рабочего (рисунок 10). Здесь значения, которые мы не хотим менять, остаются пустыми (о чём и напоминает заголовок страницы). Кнопка «Уволить» отрежет доступ соответствующего официанта к приложению.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 10 – Страница с формой редактирования данных официанта

Страница «Закуски» продемонстрирована на рисунках 11–12. Здесь блюда сортируются по четырём типам: горячее, десерты, напитки и салаты. В каждой строке таблицы данных дополнительных блюд располагается выделенная жирным шрифтом кнопка «Убрать из меню», по нажатию на которую соответствующий компонент меню становится не доступным для заказа. Восстановить его можно на странице на странице с неактивными компонентами (рисунок 13), которая откроется по клику на кнопку «Восстановить».

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 11 – Начало страницы с имеющимися в меню закусками

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 12 – Конец страницы с имеющимися в меню закусками

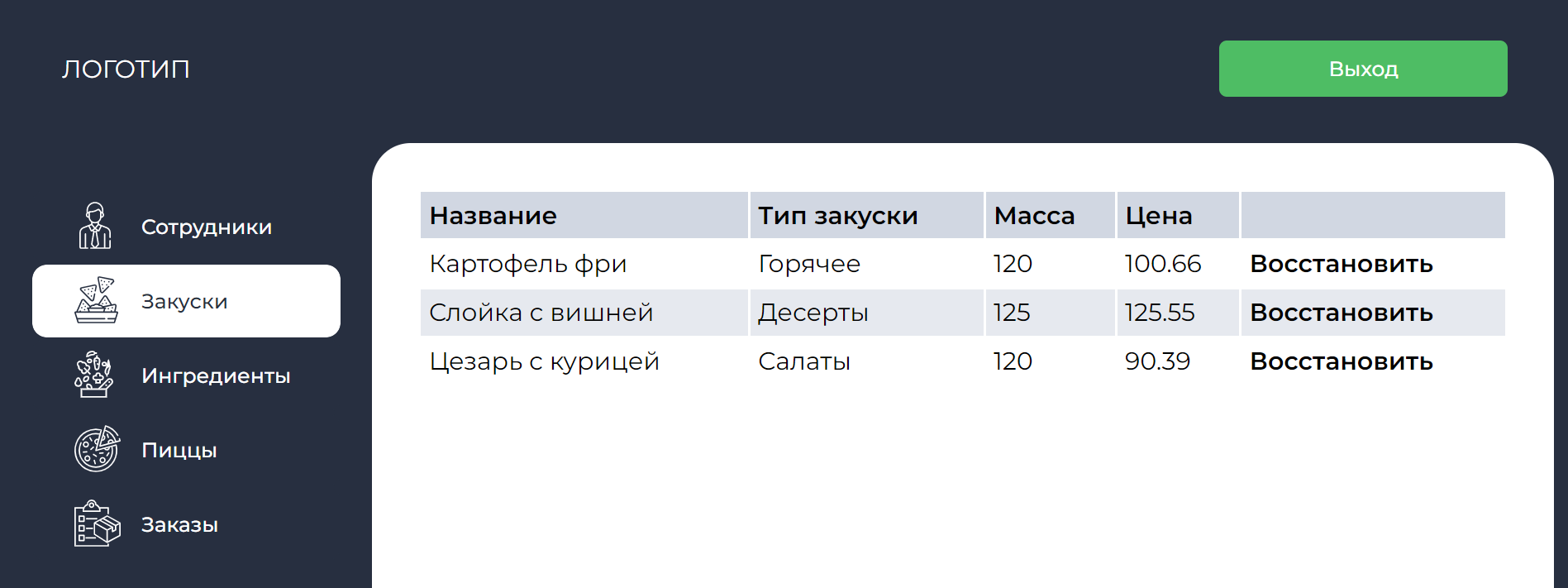


Рисунок 13 – Страница с отсутствующими в меню закусками

На странице закусок располагается кнопка «Добавить», которая перенаправит администратора на страницу с формой (рисунок 14).

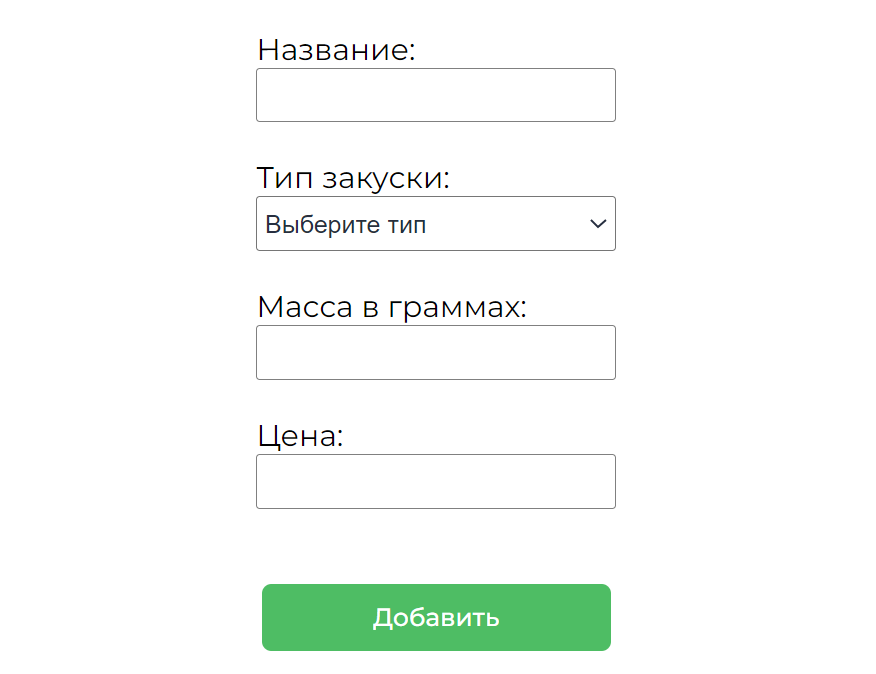


Рисунок 14 – Форма добавления закусок в меню

Страница «Ингредиенты» содержит приблизительно аналогичный странице с закусками функционал и оформление, сущность «Ингредиент» отличается от сущности «Дополнительное блюдо» отсутствием разбиения по типам.

На страницу «Пиццы» (рисунок 15) выводятся пиццы из меню с их составом и двумя кнопками-ссылками у каждой. По клику на кнопку «Подробнее» администратор окажется на странице с размерами соответствующей пиццы (рисунок 16), а по клику на «Убрать из меню» пицца становится не доступной для заказа. Восстановить её можно, перейдя на страницу восстановления (рисунок 17) по нажатию на кнопку «Восстановить».

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 – Страница с имеющимися в меню пиццами

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 16 – Страница с размерами определённой пиццы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 – Страница с отсутствующими в меню пиццами

При нажатии на кнопку «Добавить» администратор оказывается на странице формы добавления новой пиццы (рисунок 18), куда записываются масса и цена для каждой пиццы.

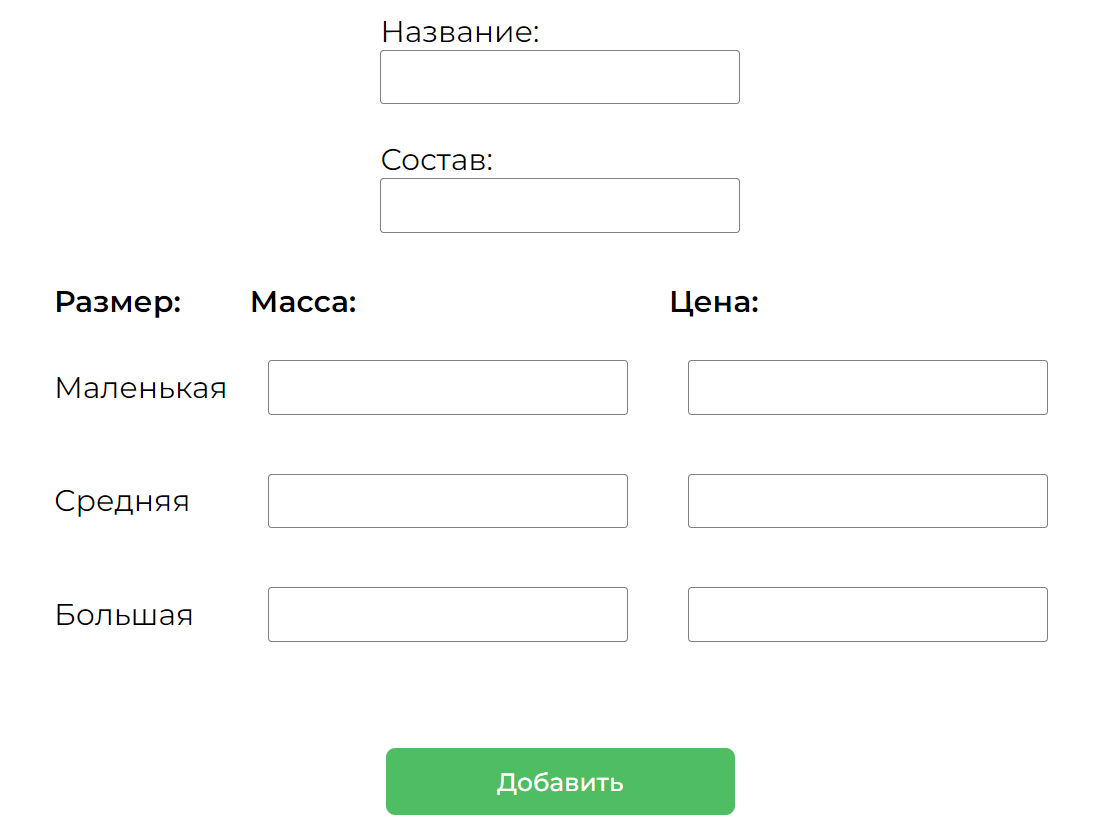


Рисунок 18 – Форма добавления пиццы

Страница «Заказы» (рисунок 19) содержит информацию о заказах клиентов. В каждой строчке столбца есть кнопка со знаком «+», по нажатию на которую можно увидеть детали заказа. На 20 рисунке демонстрируется состояние после клика.

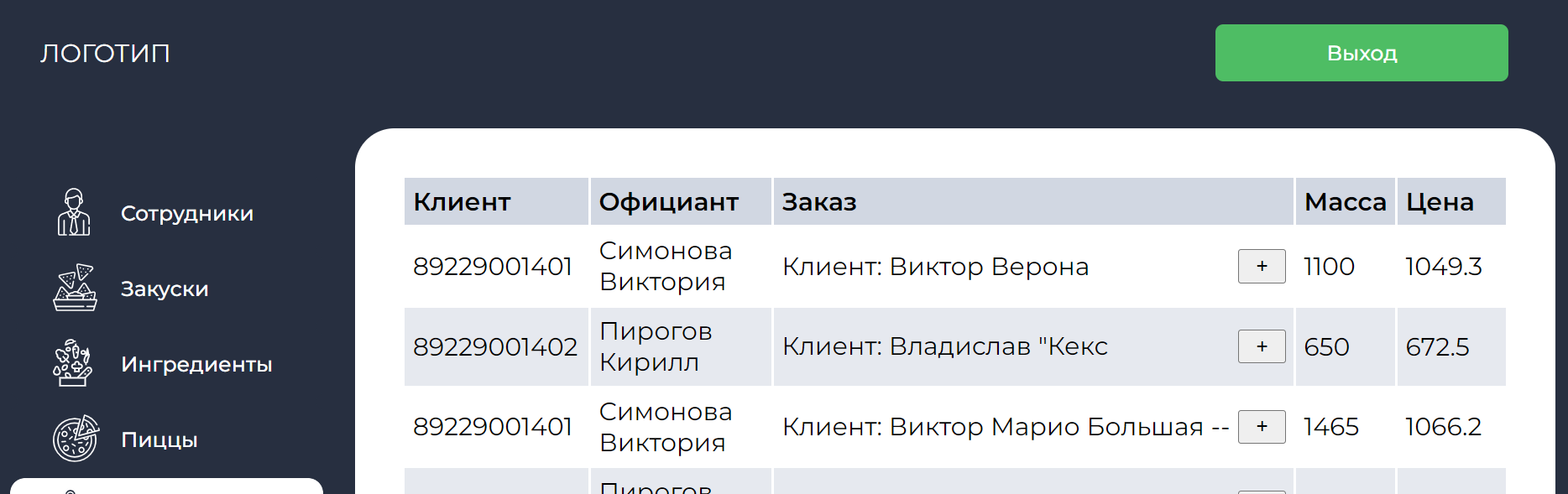


Рисунок 19 – Страница с историей заказов

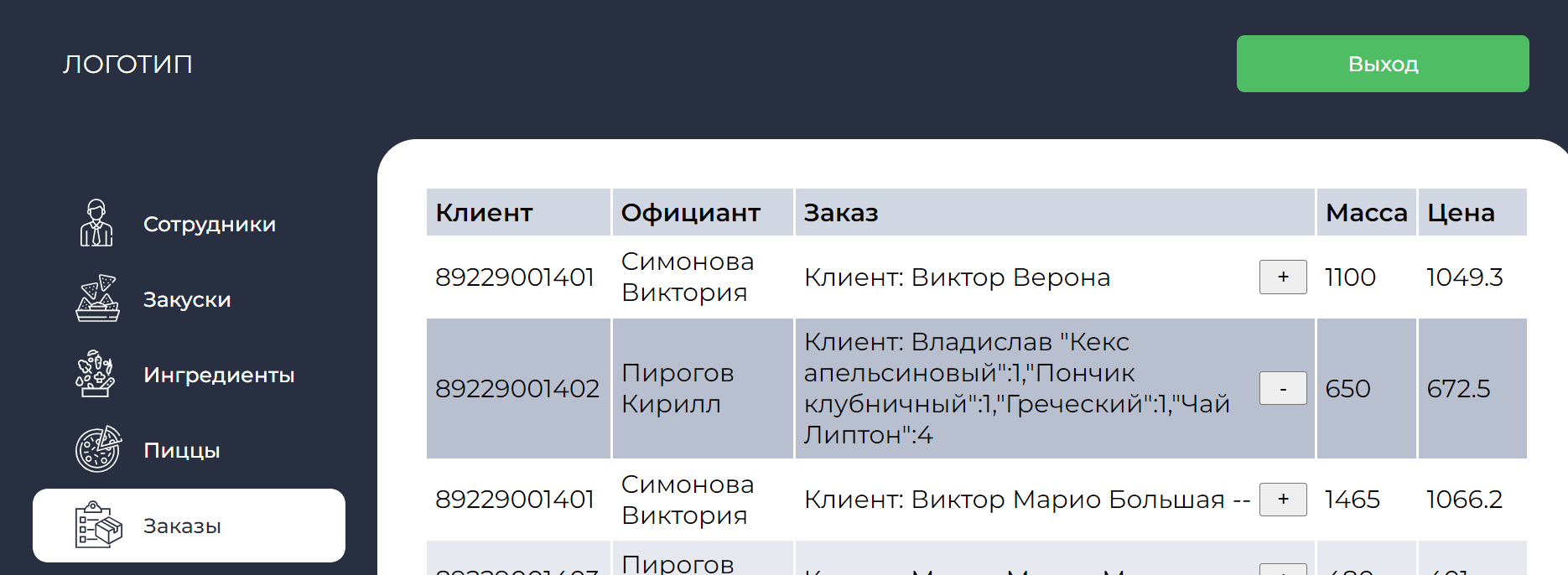


Рисунок 20 – Страница с историей заказов после нажатия кнопки «+»

Если введённые в форму входа значения соответствуют хранящимся в базе данным сотрудника, то пользователь увидит страницу приветствия официанта (рисунок 21).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21 – Страница приветствия официанта

На этой странице помимо имени официанта и указания номера пиццерии, в которой он работает, располагается кнопка «Начать работу». По клику на неё авторизованный официант переходит на страницу ввода данных клиента (рисунок 22). Если гость впервые делает заказ в пиццерии, то необходимо ввести его телефон и имя, а если он уже зарегистрирован, то будет достаточно только телефона – о чём и напомнит подсказка перед формой. При нажатии на кнопку «Добавить» данные клиента можно будет связать с выбранными блюдами.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 – Страница идентификации клиента

На следующей за идентификацией клиента странице (рисунок 23) содержится весь функционал, необходимый официанту для работы, – ключевые кнопки для оформления заказа. Информацию заказа можно посмотреть, кликнув на «Заказ», тогда слева появится панель с актуальными данными.

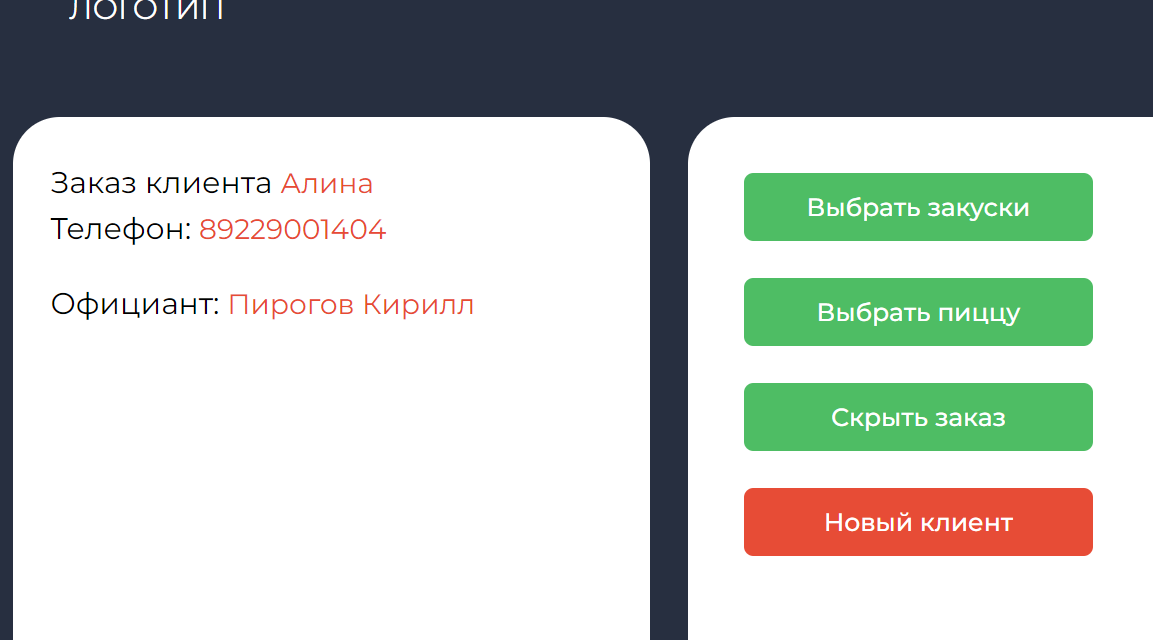


Рисунок 23 – Страница с функционалом оформления заказа и боковой панелью актуальных данных

Добавить закуски можно, перейдя на новую страницу (рисунок 24) по кнопке «Выбрать закуски». Здесь отображаются дополнительные блюда, отсортированные по типам, доступно динамическое редактирование выбранных блюд, осуществляемое с помощью кнопок «+» и «-», цветом подсвечивается количество выбранной закуски, отличное от нуля, формируется итоговая масса и цена за закуски. Функционал страницы разработан на языке javascript - языка, незаменимого для web-разработки из-за его тесной взаимосвязи с HTML- и CSS-элементами. Листинг кода приведён в приложении Г.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 24 – Страница выбора закусок

После клика на кнопку «Готово» официант вернётся на страницу с функционалом оформления заказа, а добавленные закуски отобразятся на боковой панели (рисунок 25). На ту же страницу вернёт и кнопка «Назад», но изменения в структуре заказа не будут сохранены.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25 – Страница с функционалом оформления заказа и боковой панелью актуальных данных после добавления закусок в заказ

Добавить пиццу в заказ можно, перейдя на новую страницу (рисунок 26) по кнопке «Выбрать закуски». Здесь можно выбрать пиццу определённого размера, зная её цену и массу.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 26 – Страница добавления ингредиентов к пицце

После клика на кнопку «Добавить в заказ» официант оказывается на странице, на которой можно добавить ингредиенты в пиццу (рисунок 27).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 27 – Страница с зафиксированными на боковой панели актуальными данными

Аналогичным странице закусок образом здесь работают кнопки «Готово» и «Назад». На боковой панели с актуальными данными заказа (рисунок 28) при наличии в меню и закусок, и пицц появляется строчка «Итого», в которой фиксируется итоговая сумма заказа.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 28 – Страница с функционалом оформления заказа и боковой панелью актуальных данных после добавления закусок и пиццы в заказ

Если в заказе есть одна или более пицц, то по кнопке «Выбрать пиццу» официант попадает на страницу с имеющимися в заказе пиццами (рисунок 29). Здесь можно добавить в соответствующую пиццу дополнительные ингредиенты по кнопке «Внести изменения» или убрать её из заказа. А по клику на «Добавить пиццу» официант оказывается на странице, уже продемонстрированной ранее на рисунке 26.

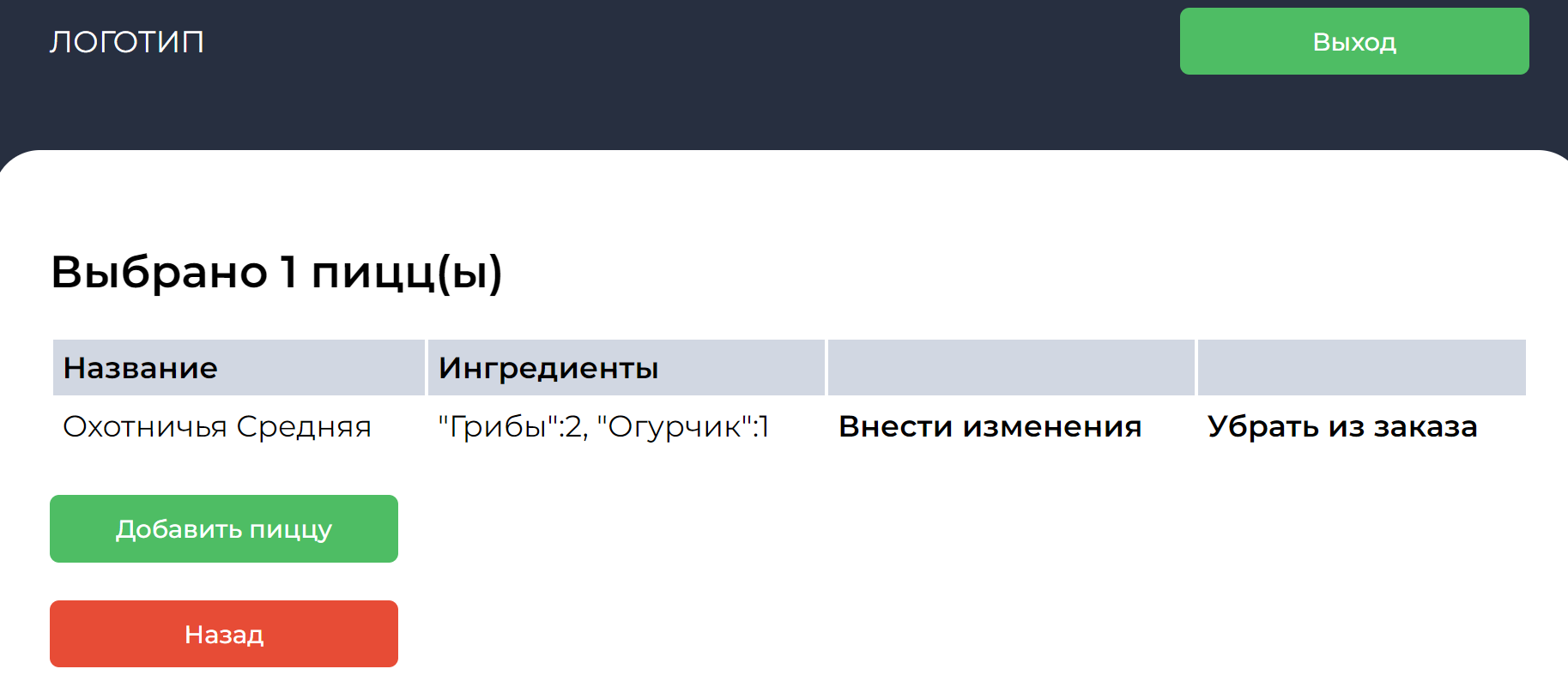


Рисунок 29 – Страница с выбранными пиццами

Внизу слева на рисунке 28 под значениями заказа клиента располагаются две кнопки: «Готово» и «Сбросить всё». Зелёная кнопка «Готово» оправит заказ в базу данных, очистит значения заказа и предложит официанту обслужить нового клиента на странице, изображённой на рисунке 22. Красная кнопка «Сбросить всё» очистит все выбранные клиентом блюда в заказе, но при этом официант продолжит его обслуживание: процесс оформления вернётся к состоянию, показанном на рисунке 23.

Кнопка «Новый клиент» (рисунок 28) отправит официанта к странице ввода данных для обслуживания нового клиента (рисунок 22).

# Заключение

В ходе выполнения курсовой работы поставленные задачи были достигнуты. Была исследована предметная область «Пиццерия» и проанализированы протекающие в ней процессы, подлежащие автоматизации. Были определены классы и отношения между сущностями, сгенерирована база данных. Было разработано web-приложение с применением объектно-ориентированного подхода и языка программирования C#. Для реализации классов была использована библиотека система Entity Framework, а интерфейс реализован с помощью технологии ASP.NET.

Приложение А

(обязательное)

Листинг программного кода реализации классов предметной области

#region Addish

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class Addish

{

public int AddishId { get; set; }

[Display(Name = "Название:")]

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Введите название!")]

public string AddishName { get; set; }

[Display(Name = "Тип закуски:")]

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Выберите!")]

public string TypeAddish { get; set; }

[Display(Name = "Масса в граммах:")]

[Required(ErrorMessage = "Введите массу!")]

public int? Mass { get; set; }

[Display(Name = "Цена:")]

[Required(ErrorMessage = "Введите стоимость!")]

public decimal? Price { get; set; }

public bool IsActive { get; set; } = true;

}

}

#endregion

#region Client

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class Client

{

[Key]

[Required(ErrorMessage = "Вы не ввели телефон!")]

[Display(Name = "Телефон:")]

public long Telephone { get; set; }

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Display(Name = "Имя:")]

public string? ClientName { get; set; }

public ICollection<ClientOrder> ClientOrders { get; set; }

}

}

#endregion

#region ClientOrder

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class ClientOrder

{

public int ClientOrderId { get; set; }

public long? Telephone { get; set; }

public Client Client { get; set; }

public int? EmployeeId { get; set; }

public Employee Employee { get; set; }

public string? NameWaiter { get; set; }

[Required]

public string TextOrder { get; set; }

[Required]

public int LastMass { get; set; }

[Required]

public double LastPrice { get; set; }

}

}

#endregion

#region Employee

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace WebApplication.Data.Entity

{

public class Employee

{

public int EmployeeId { get; set; }

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Вы не ввели фамилию!")]

[Display(Name = "Фамилия:")]

public string Lname { get; set; }

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Вы не ввели имя!")]

[Display(Name = "Имя:")]

public string Fname { get; set; }

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Display(Name = "Отчество:")]

public string? Mname { get; set; }

public int? RoleId { get; set; }

public Role Role { get; set; }

public int? PizzeriaId { get; set; }

public Pizzeria Pizzeria { get; set; }

[MaxLength(20)]

[Required(ErrorMessage = "Вы не ввели логин!")]

[Display(Name = "Логин:")]

public string Login { get; set; }

//[MaxLength(20)]

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Вы не ввели пароль!")]

[UIHint("Password")]

[Display(Name = "Пароль:")]

public string Password { get; set; }

public bool IsActive { get; set; } = true;

public ICollection<ClientOrder> ClientOrders { get; set; }

}

}

#endregion

#region Ingredient

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class Ingredient

{

public int IngredientId { get; set; }

[Display(Name = "Название:")]

[StringLength(20, ErrorMessage = "Недопустимое количество символов!")]

[Required(ErrorMessage = "Введите название!")]

public string IngredientName { get; set; }

[Display(Name = "Масса в граммах:")]

[Required(ErrorMessage = "Введите массу!")]

public int? Mass { get; set; }

[Display(Name = "Цена:")]

[Required(ErrorMessage = "Введите стоимость!")]

public decimal? Price { get; set; }

public bool IsActive { get; set; } = true;

}

}

#endregion

#region PizzaInMenu

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class PizzaInMenu

{

public int PizzaInMenuId { get; set; }

public string PizzaInMenuName { get; set; }

public string Components { get; set; }

public bool IsActive { get; set; } = true;

public ICollection<PizzaSize> PizzaSize { get; set; }

}

}

#endregion

#region PizzaSize

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace WebApplication.Data.Entity

{

public class PizzaSize

{

public int PizzaSizeId { get; set; }

public int Mass { get; set; }

public decimal Price { get; set; }

//public int? SizeNum { get; set; }

public string SizeName { get; set; }

public int? PizzaInMenuId { get; set; }

public PizzaInMenu PizzaInMenu { get; set; }

}

}

#endregion

#region Pizzeria

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

namespace WebApplication.Data.Entity

{

public class Pizzeria

{

public int PizzeriaId { get; set; }

[Required] public string Address { get; set; }

public ICollection<Employee> Employees { get; set; }

}

}

#endregion

#region Role

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

namespace WebApplication.Data.Entity

{

public class Role

{

public int RoleId { get; set; }

[MaxLength(20)] [Required] public string RoleName { get; set; }

public ICollection<Employee> Employees { get; set; }

}

}

#endregion

Приложение Б

(обязательное)

Листинг программного кода ApplicationContext

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Npgsql;

using WebApplication.Data.Entity;

namespace WebApplication.Data

{

public class ApplicationContext : DbContext

{

public DbSet<Pizzeria> Pizzerias { get; set; }

public DbSet<Role> Roles { get; set; }

public DbSet<Employee> Employees { get; set; }

public DbSet<Addish> Addishes { get; set; }

public DbSet<Ingredient> Ingredients { get; set; }

public DbSet<PizzaInMenu> Pizzas { get; set; }

public DbSet<PizzaSize> PizzaSizes { get; set; }

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

//public DbSet<ChosenAddish> ChosenAddishes { get; set; }

//public DbSet<ChosenPizza> ChosenPizzas { get; set; }

public DbSet<ClientOrder> ClientOrders { get; set; }

public ApplicationContext(DbContextOptions<ApplicationContext> options)

: base(options)

{

//Database.EnsureDeleted();

//Database.EnsureCreated();

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

}

}

}

Приложение В

(обязательное)

Листинг миграций, сгенерированных Entity Framework

using Microsoft.EntityFrameworkCore.Migrations;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace WebApplication.Migrations

{

public partial class create\_table : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Addishes",

columns: table => new

{

AddishId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

AddishName = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

Mass = table.Column<int>(type: "integer", nullable: false),

Price = table.Column<decimal>(type: "numeric", nullable: false),

IsActive = table.Column<bool>(type: "boolean", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Addishes", x => x.AddishId);

});

migrationBuilder.CreateTable(

name: "Clients",

columns: table => new

{

Telephone = table.Column<long>(type: "bigint", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

ClientName = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_Clients", x => x.Telephone);

});

migrationBuilder.CreateTable(

name: "Ingredients",

columns: table => new

{

IngredientId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

IngredientName = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

Mass = table.Column<int>(type: "integer", nullable: false),

Price = table.Column<decimal>(type: "numeric", nullable: false),

IsActive = table.Column<bool>(type: "boolean", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Ingredients", x => x.IngredientId);

});

migrationBuilder.CreateTable(

name: "Pizzas",

columns: table => new

{

PizzaInMenuId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

PizzaInMenuName = table.Column<string>(type: "text", nullable: false),

IsActive = table.Column<bool>(type: "boolean", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Pizzas", x => x.PizzaInMenuId);

});

migrationBuilder.CreateTable(

name: "Pizzerias",

columns: table => new

{

PizzeriaId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Address = table.Column<string>(type: "text", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Pizzerias", x => x.PizzeriaId);

});

migrationBuilder.CreateTable(

name: "Roles",

columns: table => new

{

RoleId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

RoleName = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Roles", x => x.RoleId);

});

migrationBuilder.CreateTable(

name: "PizzaSizes",

columns: table => new

{

PizzaSizeId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Mass = table.Column<int>(type: "integer", nullable: false),

Price = table.Column<decimal>(type: "numeric", nullable: false),

SizeName = table.Column<string>(type: "text", nullable: false),

PizzaInMenuId = table.Column<int>(type: "integer", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_PizzaSizes", x => x.PizzaSizeId);

table.ForeignKey(

name: "FK\_PizzaSizes\_Pizzas\_PizzaInMenuId",

column: x => x.PizzaInMenuId,

principalTable: "Pizzas",

principalColumn: "PizzaInMenuId");

});

migrationBuilder.CreateTable(

name: "Employees",

columns: table => new

{

EmployeeId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Lname = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

Fname = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

Mname = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: true),

RoleId = table.Column<int>(type: "integer", nullable: true),

PizzeriaId = table.Column<int>(type: "integer", nullable: true),

Login = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

Password = table.Column<string>(type: "character varying(20)", maxLength: 20, nullable: false),

IsActive = table.Column<bool>(type: "boolean", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Employees", x => x.EmployeeId);

table.ForeignKey(

name: "FK\_Employees\_Pizzerias\_PizzeriaId",

column: x => x.PizzeriaId,

principalTable: "Pizzerias",

principalColumn: "PizzeriaId");

table.ForeignKey(

name: "FK\_Employees\_Roles\_RoleId",

column: x => x.RoleId,

principalTable: "Roles",

principalColumn: "RoleId");

});

migrationBuilder.CreateTable(

name: "ClientOrders",

columns: table => new

{

ClientOrderId = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Telephone = table.Column<long>(type: "bigint", nullable: true),

ClientTelephone = table.Column<long>(type: "bigint", nullable: false),

EmployeeId = table.Column<int>(type: "integer", nullable: true),

TextOrder = table.Column<string>(type: "text", nullable: false),

LastPrice = table.Column<int>(type: "integer", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_ClientOrders", x => x.ClientOrderId);

table.ForeignKey(

name: "FK\_ClientOrders\_Clients\_ClientTelephone",

column: x => x.ClientTelephone,

principalTable: "Clients",

principalColumn: "Telephone",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_ClientOrders\_Employees\_EmployeeId",

column: x => x.EmployeeId,

principalTable: "Employees",

principalColumn: "EmployeeId");

});

migrationBuilder.CreateIndex(

name: "IX\_ClientOrders\_ClientTelephone",

table: "ClientOrders",

column: "ClientTelephone");

migrationBuilder.CreateIndex(

name: "IX\_ClientOrders\_EmployeeId",

table: "ClientOrders",

column: "EmployeeId");

migrationBuilder.CreateIndex(

name: "IX\_Employees\_PizzeriaId",

table: "Employees",

column: "PizzeriaId");

migrationBuilder.CreateIndex(

name: "IX\_Employees\_RoleId",

table: "Employees",

column: "RoleId");

migrationBuilder.CreateIndex(

name: "IX\_PizzaSizes\_PizzaInMenuId",

table: "PizzaSizes",

column: "PizzaInMenuId");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "Addishes");

migrationBuilder.DropTable(

name: "ClientOrders");

migrationBuilder.DropTable(

name: "Ingredients");

migrationBuilder.DropTable(

name: "PizzaSizes");

migrationBuilder.DropTable(

name: "Clients");

migrationBuilder.DropTable(

name: "Employees");

migrationBuilder.DropTable(

name: "Pizzas");

migrationBuilder.DropTable(

name: "Pizzerias");

migrationBuilder.DropTable(

name: "Roles");

}

}

}

Приложение Г

(обязательное)

Ключевые фрагменты кода, отвечающие за функционал программы

using Microsoft.AspNetCore.Mvc;

using WebApplication.Data;

using System.Linq;

using WebApplication.Data.Entity;

using WebApplication.Data.EntityHelp;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Http;

using System.Collections.Generic;

using System.Web;

namespace WebApplication.Controllers

{

public class HomeController : Controller

{

ApplicationContext db;

public HomeController(ApplicationContext context)

{

db = context;

}

static public int idPizzaInUser;

static public int idUser;

public static List<string> staticPizzasList = new List<string>();

public static List<string> staticIngredsList = new List<string>();

public static List<int> staticMassList = new List<int>();

public static List<decimal> staticPriceList = new List<decimal>();

public void Cleaner()

{

staticMassList.Clear();

staticPriceList.Clear();

staticPizzasList.Clear();

staticIngredsList.Clear();

HttpContext.Session.Remove("clientAddishNames");

HttpContext.Session.Remove("clientAddishPrice");

HttpContext.Session.Remove("clientAddishMass");

HttpContext.Session.Remove("clientPizza");

HttpContext.Session.Remove("clientPizzaMass");

HttpContext.Session.Remove("clientPizzaPrice");

//HttpContext.Session.Clear();

}

public void AddInSession(string totalPizzaName, string? totalOrder, decimal totalPrice, int totalMass)

{

staticPizzasList.Add(totalPizzaName);

staticMassList.Add(totalMass);

staticPriceList.Add(totalPrice);

if (String.IsNullOrEmpty(totalOrder))

staticIngredsList.Add("Без доп. ингредиентов");

else

staticIngredsList.Add(totalOrder);

//int getId = int.Parse(totalPizzaName.Split(' ')[0]);

string containerPizzas = String.Join(", ", staticPizzasList.ToArray());

string containerIngreds = String.Join(", ", staticIngredsList.ToArray());

string containerMass = staticMassList.ToArray().Sum().ToString();

string containerPrice = staticPriceList.ToArray().Sum().ToString();

HttpContext.Session.SetString("clientPizza", $"{containerPizzas} + {containerIngreds} ");

HttpContext.Session.SetString("clientPizzaMass", containerMass);

HttpContext.Session.SetString("clientPizzaPrice", containerPrice);

}

public IActionResult Login\_app()

{

return View("~/Views/Home/Login.cshtml");

}

[HttpPost]

public async Task<IActionResult> Login\_app(string login, string password)

{

Employee employee = await db.Employees.Where(e => e.IsActive == true)

.FirstOrDefaultAsync(e => e.Login == login && e.Password == password);

if (employee == null)

{

ModelState.AddModelError(nameof(login), " ");

ModelState.AddModelError(nameof(password), "Пользователь не найден...");

}

if (ModelState.IsValid)

{

ViewBag.NameOne = employee.Fname;

ViewBag.NameTwo = employee.Mname;

HttpContext.Session.SetString("employeeName", $"{employee.Lname} {employee.Fname}");

idUser = employee.EmployeeId;

if (employee.PizzeriaId != null) //для хранения глобальной переменной - id пиццерии у user

{

idPizzaInUser = int.Parse(employee.PizzeriaId.ToString());

ViewBag.PizzIdBag = idPizzaInUser;

}

if (employee.RoleId == 1)

return View("~/Views/Home/HomeAdmin/HomeAdmin.cshtml", employee);

else

return View("~/Views/Home/HomeWaiter/HomeWaiter.cshtml", employee);

}

else

return View("~/Views/Home/Login.cshtml");

}

public IActionResult Logout() //функцию очистки вставить

{

Cleaner();

return View("~/Views/Home/Login.cshtml");

}

#region Empls

[HttpPost]

public async Task<IActionResult> Empls\_panel()

{

// админу должны быть видны только работники его пиццерии

return View("~/Views/Home/HomeAdmin/Empls\_panel.cshtml",

await db.Employees

.Where(e => e.IsActive == true && e.Pizzeria.PizzeriaId.Equals(idPizzaInUser)

&& e.Role.RoleId.Equals(2)).OrderBy(e => e.Lname).ToListAsync());

}

[HttpPost]

public IActionResult Empls\_panel\_form()

{

return View("~/Views/Home/HomeAdmin/Forms/Empls\_form.cshtml");

}

[HttpPost]

public async Task<IActionResult> Add\_new\_empl(Employee model)

{

ModelState.Remove("Role");

ModelState.Remove("Pizzeria");

ModelState.Remove("ClientOrders");

if (await db.Employees.Where(e => e.Login == model.Login).FirstOrDefaultAsync() != null)

{

ModelState.AddModelError(nameof(model.Login), "Такой логин уже есть!");

}

if (ModelState.IsValid)

{

Employee employee = new Employee

{

Lname = model.Lname,

Fname = model.Fname,

Mname = model.Mname,

Login = model.Login,

Password = model.Password,

Pizzeria = await db.Pizzerias.Where(p => p.PizzeriaId == idPizzaInUser).FirstAsync(),

Role = await db.Roles.Where(r => r.RoleName == "Официант").FirstAsync(),

IsActive = model.IsActive

};

db.Employees.Add(employee);

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Empls\_panel.cshtml",

await db.Employees

.Where(e => e.IsActive == true && e.Pizzeria.PizzeriaId.Equals(idPizzaInUser)

&& e.Role.RoleId.Equals(2)).OrderBy(e => e.Lname).ToListAsync());

}

else

{

return View("~/Views/Home/HomeAdmin/Forms/Empls\_form.cshtml");

}

}

public async Task<IActionResult> Empls\_reactive(int? id)

{

if (id != null)

{

Employee employee = await db.Employees.FirstAsync(e => e.EmployeeId == id);

if (employee != null)

{

if (employee.IsActive == true)

employee.IsActive = false;

else employee.IsActive = true;

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Empls\_panel.cshtml",

await db.Employees

.Where(e => e.IsActive == true && e.Pizzeria.PizzeriaId.Equals(idPizzaInUser)

&& e.Role.RoleId.Equals(2)).OrderBy(e => e.Lname).ToListAsync());

}

}

return NotFound();

}

public async Task<IActionResult> Empls\_edit\_page(int? id)

{

if (id != null)

{

Employee employee = await db.Employees.FirstAsync(e => e.EmployeeId == id);

//re

ViewBag.EmplId = id;

ViewBag.EmplLn = employee.Lname;

ViewBag.EmplFn = employee.Fname;

ViewBag.EmplMn = employee.Mname;

ViewBag.EmplLo = employee.Login;

ViewBag.EmplPa = employee.Password;

return View("~/Views/Home/HomeAdmin/Forms/Empls\_re.cshtml");

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Empls\_edit(EmployeeRe model)

{

if (model.EmployeeId != null)

{

Employee employee = await db.Employees.FirstAsync(e => e.EmployeeId == model.EmployeeId);

if (model.Lname != null || model.Fname != null || model.Mname != null ||

model.Password != null || model.Login != null)

{

if (model.Lname != null) employee.Lname = model.Lname;

if (model.Fname != null) employee.Fname = model.Fname;

if (model.Mname != null) employee.Mname = model.Mname;

if (model.Password != null) employee.Password = model.Password;

if (model.Login != null)

{

if (db.Employees.Where(e => e.Login == model.Login).FirstOrDefault() != null)

ModelState.AddModelError(nameof(model.Login), "Такой логин уже есть!");

else employee.Login = model.Login;

}

if (ModelState.IsValid)

{

await db.SaveChangesAsync();

}

else

{

ViewBag.EmplId = model.EmployeeId;

ViewBag.EmplLn = employee.Lname;

ViewBag.EmplFn = employee.Fname;

ViewBag.EmplMn = employee.Mname;

ViewBag.EmplLo = employee.Login;

ViewBag.EmplPa = employee.Password;

return View("~/Views/Home/HomeAdmin/Forms/Empls\_re.cshtml");

}

}

}

return View("~/Views/Home/HomeAdmin/Empls\_panel.cshtml",

await db.Employees

.Where(e => e.IsActive == true && e.Pizzeria.PizzeriaId.Equals(idPizzaInUser)

&& e.Role.RoleId.Equals(2)).OrderBy(e => e.Lname).ToListAsync());

}

public async Task<IActionResult> Empls\_unactive()

{

return View("~/Views/Home/HomeAdmin/SubPage/Empls\_unactive.cshtml",

await db.Employees.Where(e => e.IsActive == false

&& e.Pizzeria.PizzeriaId.Equals(idPizzaInUser)

&& e.Role.RoleId.Equals(2)).OrderBy(e => e.Lname).ToListAsync());

}

#endregion

#region Addishes

[HttpPost]

public async Task<IActionResult> Addishes\_panel()

{

return View("~/Views/Home/HomeAdmin/Addishes\_panel.cshtml",

await db.Addishes

.Where(e => e.IsActive == true)

.OrderBy(e => e.TypeAddish)

.ToListAsync());

}

[HttpPost]

public IActionResult Addishes\_panel\_form()

{

return View("~/Views/Home/HomeAdmin/Forms/Addishes\_form.cshtml");

}

[HttpPost]

public async Task<IActionResult> Add\_new\_addish(Addish addish)

{

if (ModelState.IsValid)

{

db.Addishes.Add(addish);

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Addishes\_panel.cshtml",

await db.Addishes

.Where(e => e.IsActive == true)

.OrderBy(e => e.TypeAddish)

.ToListAsync());

}

else

{

return View("~/Views/Home/HomeAdmin/Forms/Addishes\_form.cshtml");

}

}

public async Task<IActionResult> Addishes\_reactive(int? id)

{

if (id != null)

{

Addish addish = await db.Addishes.FirstAsync(e => e.AddishId == id);

if (addish != null)

{

if (addish.IsActive == true)

addish.IsActive = false;

else addish.IsActive = true;

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Addishes\_panel.cshtml",

await db.Addishes

.Where(e => e.IsActive == true)

.OrderBy(e => e.TypeAddish)

.ToListAsync());

}

}

return NotFound();

}

public async Task<IActionResult> Addishes\_unactive()

{

return View("~/Views/Home/HomeAdmin/SubPage/Addishes\_unactive.cshtml",

await db.Addishes

.Where(e => e.IsActive == false)

.OrderBy(e => e.TypeAddish)

.ToListAsync());

}

#endregion

#region Ingredients

[HttpPost]

public async Task<IActionResult> Ingreds\_panel()

{

return View("~/Views/Home/HomeAdmin/Ingreds\_panel.cshtml",

await db.Ingredients

.Where(e => e.IsActive == true)

.OrderBy(e => e.IngredientName)

.ToListAsync());

}

[HttpPost]

public IActionResult Ingreds\_panel\_form()

{

return View("~/Views/Home/HomeAdmin/Forms/Ingreds\_form.cshtml");

}

[HttpPost]

public async Task<IActionResult> Add\_new\_ingred(Ingredient ingred)

{

//ModelState.Remove("ChosenPizzas");

if (ModelState.IsValid)

{

db.Ingredients.Add(ingred);

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Ingreds\_panel.cshtml",

await db.Ingredients

.Where(e => e.IsActive == true)

.OrderBy(e => e.IngredientName)

.ToListAsync());

}

else

{

return View("~/Views/Home/HomeAdmin/Forms/Ingreds\_form.cshtml");

}

}

public async Task<IActionResult> Ingreds\_unactive()

{

return View("~/Views/Home/HomeAdmin/SubPage/Ingreds\_unactive.cshtml",

await db.Ingredients

.Where(e => e.IsActive == false)

.OrderBy(e => e.IngredientName)

.ToListAsync());

}

public async Task<IActionResult> Ingreds\_reactive(int? id)

{

if (id != null)

{

Ingredient ingredient = await db.Ingredients.FirstAsync(e => e.IngredientId == id);

if (ingredient != null)

{

if (ingredient.IsActive == true)

ingredient.IsActive = false;

else ingredient.IsActive = true;

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Ingreds\_panel.cshtml",

await db.Ingredients

.Where(e => e.IsActive == true)

.OrderBy(e => e.IngredientName)

.ToListAsync());

}

}

return NotFound();

}

#endregion

#region Pizzas

[HttpPost]

public async Task<IActionResult> Pizzas\_panel()

{

return View("~/Views/Home/HomeAdmin/Pizzas\_panel.cshtml",

await db.Pizzas

.Where(e => e.IsActive == true)

.OrderBy(e => e.PizzaInMenuName)

.ToListAsync());

}

[HttpPost]

public IActionResult Pizzas\_panel\_form()

{

return View("~/Views/Home/HomeAdmin/Forms/Pizzas\_form.cshtml");

}

[HttpPost]

public IActionResult Add\_new\_pizza(PizzaAdd pizzaform)

{

if (db.Pizzas.Where(p => p.PizzaInMenuName == pizzaform.PizzaName).FirstOrDefault() != null)

{

ModelState.AddModelError(nameof(pizzaform.PizzaName), "Такая пицца уже есть!");

}

if (ModelState.IsValid)

{

using (var transaction = db.Database.BeginTransaction())

{

try

{

PizzaInMenu pizza = new PizzaInMenu

{

PizzaInMenuName = pizzaform.PizzaName,

Components = (string)pizzaform.Components

};

db.Pizzas.Add(pizza);

db.SaveChanges();

PizzaSize pizzaSizeSmall = new PizzaSize

{

Mass = (int)pizzaform.MassSmall,

Price = (decimal)pizzaform.PriceSmall,

SizeName = "Маленькая",

PizzaInMenu = db.Pizzas.Where(p => p.PizzaInMenuName == pizzaform.PizzaName).First()

};

PizzaSize pizzaSizeMedium = new PizzaSize

{

Mass = (int)pizzaform.MassMedium,

Price = (decimal)pizzaform.PriceMedium,

SizeName = "Средняя",

PizzaInMenu = db.Pizzas.Where(p => p.PizzaInMenuName == pizzaform.PizzaName).First()

};

PizzaSize pizzaSizeBig = new PizzaSize

{

Mass = (int)pizzaform.MassBig,

Price = (decimal)pizzaform.PriceBig,

SizeName = "Большая",

PizzaInMenu = db.Pizzas.Where(p => p.PizzaInMenuName == pizzaform.PizzaName).First()

};

db.PizzaSizes.AddRange(pizzaSizeSmall, pizzaSizeMedium, pizzaSizeBig);

db.SaveChanges();

transaction.Commit();

return View("~/Views/Home/HomeAdmin/Pizzas\_panel.cshtml",

db.Pizzas

.Where(e => e.IsActive == true)

.OrderBy(e => e.PizzaInMenuName)

.ToList());

}

catch

{

transaction.Rollback();

return View("~/Views/Home/Error\_page.cshtml");

}

}

}

else return View("~/Views/Home/HomeAdmin/Forms/Pizzas\_form.cshtml");

}

public async Task<IActionResult> Pizzas\_unactive()

{

return View("~/Views/Home/HomeAdmin/SubPage/Pizzas\_unactive.cshtml",

await db.Pizzas

.Where(e => e.IsActive == false)

.OrderBy(e => e.PizzaInMenuName)

.ToListAsync());

}

public async Task<IActionResult> Pizzas\_reactive(int? id)

{

if (id != null)

{

PizzaInMenu pizza = await db.Pizzas.FirstAsync(e => e.PizzaInMenuId == id);

if (pizza != null)

{

if (pizza.IsActive == true)

pizza.IsActive = false;

else pizza.IsActive = true;

await db.SaveChangesAsync();

return View("~/Views/Home/HomeAdmin/Pizzas\_panel.cshtml",

await db.Pizzas

.Where(e => e.IsActive == true)

.OrderBy(e => e.PizzaInMenuName)

.ToListAsync());

}

}

return NotFound();

}

public async Task<IActionResult> Pizzas\_detali(int? id)

{

if (id != null)

{

PizzaInMenu pizza = await db.Pizzas.FirstAsync(e => e.PizzaInMenuId == id);

if (pizza != null)

{

ViewBag.NamePizza = pizza.PizzaInMenuName;

return View("~/Views/Home/HomeAdmin/SubPage/Pizzas\_detali.cshtml",

await db.PizzaSizes

.Where(s => s.PizzaInMenu.PizzaInMenuId == id)

.ToListAsync());

}

}

return NotFound();

}

#endregion

[HttpPost]

public async Task<IActionResult> Orders\_panel()

{

return View("~/Views/Home/HomeAdmin/Orders\_panel.cshtml",

await db.ClientOrders

.ToListAsync());

}

#region Works

/\*------------------------ WORKS START ------------------------ \*/

[HttpPost]

public IActionResult Work\_panel()

{

return View("~/Views/Home/HomeWaiter/Startwork\_panel.cshtml");

}

[HttpPost] //страница ввода данных клиента

public async Task<IActionResult> Add\_new\_client(Client model)

{

ModelState.Remove("ClientOrders");

Client ifExist = await db.Clients.Where(c => c.Telephone == model.Telephone).FirstOrDefaultAsync();

if (ifExist == null)

{

//если клиента с таким номером нет

if (model.ClientName == null)

ModelState.AddModelError(nameof(model.ClientName), "Введите имя!");

if (ModelState.IsValid)

{

Client client = new Client

{

Telephone = model.Telephone,

ClientName = model.ClientName

};

db.Clients.Add(client);

await db.SaveChangesAsync();

HttpContext.Session.SetString("clientName", model.ClientName);

HttpContext.Session.SetString("clientTel", model.Telephone.ToString());

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

else //если при регистрации не продит проверку на валидность

return View("~/Views/Home/HomeWaiter/Startwork\_panel.cshtml");

}

else

{

//если киент с таким номером уже есть

HttpContext.Session.SetString("clientName", ifExist.ClientName);

HttpContext.Session.SetString("clientTel", ifExist.Telephone.ToString());

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

return NotFound();

}

[HttpPost] //кнопка назад

public IActionResult Order\_back()

{

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

[HttpPost]

public IActionResult New\_client() //отмена заказа и удаление сохраненных промежуточных данных

{

Cleaner();

//clientTel

HttpContext.Session.Remove("clientTel");

HttpContext.Session.Remove("clientName");

return View("~/Views/Home/HomeWaiter/Startwork\_panel.cshtml");

}

[HttpPost] //панель с дополнительными блюдами

public async Task<IActionResult> Addish\_order()

{

return View("~/Views/Home/HomeWaiter/Addish\_panel.cshtml",

await db.Addishes

.Where(e => e.IsActive == true)

.OrderBy(e => e.TypeAddish)

.ToListAsync());

}

[HttpPost] //кнопка отправки данных в session

public IActionResult Get\_addish\_order(string totalOrder, decimal totalPrice, int totalMass)

{

if (!String.IsNullOrEmpty(totalOrder))

{

HttpContext.Session.SetString("clientAddishNames", totalOrder);

HttpContext.Session.SetString("clientAddishPrice", totalPrice.ToString());

HttpContext.Session.SetString("clientAddishMass", totalMass.ToString());

}

else

{

HttpContext.Session.Remove("clientAddishNames");

HttpContext.Session.Remove("clientAddishPrice");

HttpContext.Session.Remove("clientAddishMass");

}

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

[HttpPost] //панель с пицца join размер

public async Task<IActionResult> Pizza\_order()

{

if (staticPizzasList.Count == 0)

{

List<PizzaInMenu> pizzaInMenu = await db.Pizzas.Where(p => p.IsActive == true).OrderBy(p => p.PizzaInMenuName).ToListAsync();

List<PizzaSize> pizzaSize = await db.PizzaSizes.ToListAsync();

var linq = from s in pizzaSize

join m in pizzaInMenu on s.PizzaInMenuId equals m.PizzaInMenuId

select new PizzaAndSize { PizzaSize = s, PizzaInMenu = m };

return View("~/Views/Home/HomeWaiter/Pizzas\_panel.cshtml", linq);

}

else //если мы выбралибольше одной пиццы появится панель Pizzas\_panel\_two с нашими выбранными пиццами

//там появится возможность внести изменения в набор ингредиентов

{

Dictionary<string, string> dic = new Dictionary<string, string>(); //словарь вывода

for (int i = 0; i < staticPizzasList.Count; i++)

{

dic.Add(staticPizzasList[i].Remove(0, 2), staticIngredsList[i].Trim(new char[] { '{', '}' }).Replace(",", ", "));

}

//return View("~/Views/Home/HomeWaiter/Pizzas\_panel\_two.cshtml", staticPizzasList);

return View("~/Views/Home/HomeWaiter/Pizzas\_panel\_two.cshtml", dic);

}

}

[HttpPost] //панель с пицца join размер при повторном вызове

public async Task<IActionResult> Pizza\_order\_two()

{

List<PizzaInMenu> pizzaInMenu = await db.Pizzas.Where(p => p.IsActive == true).OrderBy(p => p.PizzaInMenuName).ToListAsync();

List<PizzaSize> pizzaSize = await db.PizzaSizes.ToListAsync();

var linq = from s in pizzaSize

join m in pizzaInMenu on s.PizzaInMenuId equals m.PizzaInMenuId

select new PizzaAndSize { PizzaSize = s, PizzaInMenu = m };

return View("~/Views/Home/HomeWaiter/Pizzas\_panel.cshtml", linq);

}

public async Task<IActionResult> Pizza\_add\_order(int? id) //первый раз добавляем пиццу

{

PizzaSize pizzaSize = await db.PizzaSizes.Where(s => s.PizzaSizeId == id).FirstAsync();

PizzaInMenu pizzaInMenu = await db.Pizzas.Where(p => p.PizzaInMenuId == pizzaSize.PizzaInMenuId).FirstAsync();

ViewBag.IdPizz = pizzaSize.PizzaSizeId;

ViewBag.Name = pizzaInMenu.PizzaInMenuName;

ViewBag.Size = pizzaSize.SizeName;

ViewBag.Mass = pizzaSize.Mass;

ViewBag.Price = pizzaSize.Price;

return View("~/Views/Home/HomeWaiter/PizzaIngreds\_panel.cshtml",

await db.Ingredients

.Where(e => e.IsActive == true)

.ToListAsync());

}

public async Task<IActionResult> Pizza\_change(int? id) //изменение набора доп. ингредиентов к пицце

{

if (id != null)

{

//int.Parse(staticPizzasList[(int)id].Split(' ')[0]);

//получаем id пиццы из БД, которую мы выбрали

int idPizza = int.Parse(staticPizzasList[(int)id].Split(' ')[0]);

PizzaSize pizzaSize = await db.PizzaSizes.Where(s => s.PizzaSizeId == idPizza).FirstAsync();

PizzaInMenu pizzaInMenu = await db.Pizzas.Where(p => p.PizzaInMenuId == pizzaSize.PizzaInMenuId).FirstAsync();

ViewBag.IdPizz = pizzaSize.PizzaSizeId;

ViewBag.Name = pizzaInMenu.PizzaInMenuName;

ViewBag.Size = pizzaSize.SizeName;

ViewBag.Mass = pizzaSize.Mass;

ViewBag.Price = pizzaSize.Price;

ViewBag.IdInStatic = id;

ViewBag.Ingreds = staticIngredsList[(int)id];

return View("~/Views/Home/HomeWaiter/PizzaIngreds\_panel\_two.cshtml",

await db.Ingredients

.Where(e => e.IsActive == true)

.ToListAsync());

}

return NotFound();

}

[HttpPost] //добавляем пиццы в session

public async Task<IActionResult> Get\_pizza\_order(string totalPizzaName, string? totalOrder, decimal totalPrice, int totalMass)

{

AddInSession(totalPizzaName, totalOrder, totalPrice, totalMass);

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

[HttpPost] //замена пиццы

public async Task<IActionResult> Get\_pizza\_order\_two(int id, string totalPizzaName, string? totalOrder, decimal totalPrice, int totalMass)

{

staticPizzasList.RemoveAt(id);

staticIngredsList.RemoveAt(id);

staticPriceList.RemoveAt(id);

staticMassList.RemoveAt(id);

AddInSession(totalPizzaName, totalOrder, totalPrice, totalMass);

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

public IActionResult Pizza\_delete(int id)

{

staticPizzasList.RemoveAt(id);

staticIngredsList.RemoveAt(id);

staticPriceList.RemoveAt(id);

staticMassList.RemoveAt(id);

string containerPizzas = String.Join(", ", staticPizzasList.ToArray());

string containerIngreds = String.Join(", ", staticIngredsList.ToArray());

string containerMass = staticMassList.ToArray().Sum().ToString();

string containerPrice = staticPriceList.ToArray().Sum().ToString();

HttpContext.Session.SetString("clientPizza", $"{containerPizzas} + {containerIngreds} ");

HttpContext.Session.SetString("clientPizzaMass", containerMass);

HttpContext.Session.SetString("clientPizzaPrice", containerPrice);

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

public async Task<IActionResult> Done\_order()

{

if (staticPizzasList.Count() > 0 || HttpContext.Session.GetString("clientAddishNames") != null)

{

string text = "Клиент: " + HttpContext.Session.GetString("clientName") + "\n";

double price = 0;

int mass = 0;

if (staticPizzasList.Count() > 0)

{

price += Double.Parse(HttpContext.Session.GetString("clientPizzaPrice"));

mass += Int32.Parse(HttpContext.Session.GetString("clientPizzaMass"));

for (int i = 0; i < staticPizzasList.Count(); i++)

text += staticPizzasList[i].Substring(2) + " --> " + staticIngredsList[i].Trim(new char[] { '{', '}' }) + "\n";

}

if (HttpContext.Session.GetString("clientAddishNames") != null)

{

price += Double.Parse(HttpContext.Session.GetString("clientAddishPrice"));

mass += Int32.Parse(HttpContext.Session.GetString("clientAddishMass"));

text += HttpContext.Session.GetString("clientAddishNames").Trim(new char[] { '{', '}' }) + "\n";

}

Employee employee = await db.Employees.Where(e => e.EmployeeId == idUser).FirstAsync();

ClientOrder clientOrder = new ClientOrder

{

Telephone = Int64.Parse(HttpContext.Session.GetString("clientTel")),

Client = await db.Clients.Where(c => c.Telephone == Int64.Parse(HttpContext.Session.GetString("clientTel"))).FirstAsync(),

Employee = employee,

NameWaiter = $"{employee.Lname} {employee.Fname}",

TextOrder = text,

LastMass = mass,

LastPrice = price

};

db.ClientOrders.Add(clientOrder);

await db.SaveChangesAsync();

return View("~/Views/Home/HomeWaiter/Order\_done.cshtml");

}

else return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

public IActionResult Clear\_order()

{

Cleaner();

return View("~/Views/Home/HomeWaiter/Order\_panel.cshtml");

}

/\*------------------------ WORKS END ------------------------ \*/

#endregion

}

}

Приложение Д

(справочное)

Библиографический список

1. Entity Framework – Microsoft Docs [электронный ресурс] // Последнее обновление 29.10.2021 URL: <https://docs.microsoft.com/en-us/aspnet/entity-framework> (дата обращения 19.11.2021).

2. Руководство в Entity Framework [электронный ресурс] // Последнее обновление 19.10.2021 URL: <https://metanit.com/sharp/efcore> (дата обращения 19.11.2021).

3. Документация PostgreSQL: Компания Postgres Professional [электронный ресурс] // Последнее обновление 19.10.2021 URL: <https://postgrespro.ru/docs> (дата обращения 20.11.2021).

4. Руководство по ASP.NET [электронный ресурс] // Последнее обновление 06.02.2022 URL: <https://metanit.com/sharp/aspnet6/> (дата обращения 07.01.2022).