**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184772655)

[1.1. Введение в предметную область 5](#_Toc184772656)

[1.2 Анализ предметной области 7](#_Toc184772657)

[2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 9](#_Toc184772658)

[2.1 Общее назначение системы 9](#_Toc184772659)

[2.2 Требования к функциям, выполняемым системой 10](#_Toc184772660)

[2.3 План тестирования 14](#_Toc184772661)

[3. Реализация проекта системы 15](#_Toc184772662)

[3.1 Описание среды разработки (Ide PyCharm) 16](#_Toc184772663)

[3.1.1 Выбор и описание программных инструментов. 16](#_Toc184772664)

[3.1.2 Обоснование выбора инструментарий разработки 16](#_Toc184772665)

[3.2 Разработка программного модуля 17](#_Toc184772666)

[3.2.1 Реализация пользовательского интерфейса программы 18](#_Toc184772667)

[3.2.2. Описание кодом функциональных узлов модуля 19](#_Toc184772668)

[3.2.3 Результаты работы и тестирования 23](#_Toc184772669)

[ЗАКЛЮЧЕНИЕ 25](#_Toc184772670)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc184772671)

[Приложение 1 28](#_Toc184772672)

[Приложение 2 29](#_Toc184772673)

[Приложение 3 34](#_Toc184772674)

[Приложение 4 35](#_Toc184772675)

# **ВВЕДЕНИЕ**

Сложно представить современный технологический мир без ресторанов и кафе, которые используют информационные системы для управления своим ассортиментом. Заведения общественного питания, в частности, не являются исключениями. Система учета ассортимента блюд позволяет владельцам и персоналу эффективно управлять предложением, отслеживать наличие ингредиентов, а также легко адаптировать меню в зависимости от спроса.

Цель курсового проекта «Разработка программного модуля "Учёт ассортимента блюд для общепита"» заключается в создании программного модуля, который упростит управление ассортиментом блюд в заведениях общественного питания. Модуль должен быть интуитивно понятным и удобным в использовании, что позволит сотрудникам легко взаимодействовать с системой и оперативно управлять предложением. Такая разработка поможет заведениям быстрее адаптироваться к изменениям на рынке, оптимизировать процессы и повысить качество обслуживания клиентов.

Задачи курсового проекта:

1. Проанализировать предметную область
2. Проанализировать готовые решения
3. Подготовить ТЗ
4. Подготовить план тестирования
5. Обосновать выбор инструментов и средств разработки
6. Описать реализацию ТЗ
7. Выполнить тестирование

Объектом исследования является модуль, направленный на оптимизацию процессов взаимодействия между клиентами и заведениями общественного питания, улучшение качества обслуживания и повышение уровня удовлетворенности клиентов.

Цели автоматизации:

1. Ускорение процесса заказа и подачи блюд.

2. Повышение качества обслуживания клиентов.

3. Снижение вероятности ошибок в заказах.

Предметом исследования является учет создания, хранения и обработки информации по ассортименту блюд в заведениях общественного питания. Важно рассмотреть разработку модуля, который будет направлен на оптимизацию взаимодействия между клиентами и персоналом. Простота и доступность данного модуля помогут ускорить процесс обработки заказов, сделав его комфортным для всех клиентов.

Данная тема курсового проекта актуальна, так как разработка программного модуля для учета ассортимента блюд в общественном питании способствует развитию бизнеса и значительно улучшает привлечение клиентов благодаря своим удобным функциональным возможностям.

Таким образом, разработка модуля для системы учета ассортимента блюд в заведениях общественного питания поможет в дальнейшем развитии бизнеса, привлечении клиентов, а также улучшит процесс обслуживания.

**1.** **ОПИСАНИЕ ПРЕДМЕНТНОЙ ОБЛАСТИ**

## 1.1. Введение в предметную область

Разработка программного модуля «Учёт ассортимента блюд для общепита» представляет собой проект, направленный на автоматизацию управления ассортиментом блюд в заведениях общественного питания. Данный модуль обеспечивает удобный интерфейс для администраторов и шеф-поваров, позволяя им легко добавлять, изменять и удалять позиции меню.

Программный модуль позволяет пользователям управлять разнообразными блюдами, включая различные категории, ингредиенты, размеры порций и цены. Система может включать функционал для отслеживания наличия ингредиентов на складе, что поможет в планировании закупок и минимизации потерь.

Клиенты могут просматривать меню через интерфейс, выбирая желаемые блюда, добавляя их в заказ и оформляя его. Оплата может осуществляться различными способами, включая электронные платежи.

Методология разработки программного модуля «Учёт ассортимента блюд для общепита» включает в себя несколько ключевых шагов:

1. Сбор требований
2. Проектирование
3. Внедрение и тестирование

Связь между этими шагами включает в себя:

1. Требования к системе
2. Анализ пользователей

Современный рынок общественного питания требует от информационных систем строгого выполнения определенных требований, таких как:

1. Удобный интерфейс
2. Данные для регистрации каждого сотрудника
3. Обеспечение безопасности данных
4. Аналитика и отчетность

Разработка программного модуля для учета ассортимента блюд для общепита подразумевает соблюдение ряда законодательных норм, касающихся безопасности, качества и соответствия стандартам ФЗ (Федеральный закон) и ГОСТам (Государственный стандарт)

1. ФЗ-149 "Об информации, информационных технологиях и о защите информации". Главный закон об информации в России. Определяет ключевые термины, говорит, что информация – это любые данные, сообщения и сведения, представляемые в любой форме.
2. ФЗ-152 "О персональных данных". Определяет требования по работе с персональными данными российских граждан, обеспечивает защиту их интересов и надлежащий уровень защиты.
3. ФЗ-187 "О внесении изменений в отдельные законодательные акты Российской Федерации в связи с совершенствованием правового регулирования отношений в сфере информационных технологий". Определяет основные принципы обеспечения безопасности, полномочия госорганов, а также права, обязанности и ответственность субъектов КИИ.

ГОСТы:

1. ГОСТ Р ИСО/МЭК 12207-2010 "Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств".
2. ГОСТ 19.701-90. "Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения".
3. ГОСТ 19.101-77 "Единая система программной документации. Виды программ и программных документов".

Есть много видов методологий разработки ПО. Есть несколько известных и часто используемых видов методологий, такие как Agile, cascade (каскадный), prototype. Для разработки данного модуля будет использовано водопадная методология

Водопадная модель разработки ПО включает несколько этапов разработки:

1. Сбор и анализ требований
2. Проектирование системы
3. Кодирование
4. Тестирование
5. Внедрение

Поддержка и обслуживание

## 1.2 Анализ предметной области

Виды компаний, в которых применяют информационные системы в области «Учёт ассортимента блюд для общепита»:

1. Рестораны и кафе
2. Столовые и кейтеринговые компании
3. Фастфуды и закусочные

Примеры информационных систем для разработки программного модуля «Учёт ассортимента блюд для общепита»:

1. Toast

Платформа для управления ресторанами, которая включает функции учёта блюд, обработки заказов и управления запасами.

Плюсы: Подходит для ресторанов, желающих оптимизировать процесс учёта блюд и повысить эффективность работы.

1. Square for Restaurants

Мощная система управления для ресторанов, предоставляющая инструменты для учёта ассортимента, работы с заказами и аналитики.

Плюсы: простота использования и возможность гибкого формирования меню.

Минусы: ограничения по интеграциям с другими системами и возможная стоимость обслуживания.

1. ChefTec

Специализированное программное обеспечение для управления рецептами и учёта запасов в сфере общественного питания.

Плюсы: подробный учёт ингредиентов, составление меню и управление затратами.

Минусы: требует времени на обучение и внедрение, а также может быть дорогостоящим для малых предприятий.

Таким образом, внедрение информационных систем в сфере «Учёт ассортимента блюд для общепита» повышает эффективность управления и улучшает клиентский сервис, что в конечном итоге приводит к росту прибыли и привлечению новых клиентов.

# **2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ**

## 2.1 Общее назначение системы

Одним из ключевых результатов внедрения разработанного модуля «Учёт ассортимента блюд для общепита» станет оптимизация управления ассортиментом и улучшение взаимодействия с клиентами. Новый программный модуль позволит ресторанам и кафе более эффективно управлять меню, что в свою очередь привлечет клиентов, ранее не имевших возможности оценить предлагаемые блюда. Внедрение данного модуля значительно повысит удобство работы для существующих и новых клиентов. Посетители смогут в любое время и из любого места ознакомиться с актуальным меню, ценами и специальными предложениями. Эти факторы приведут к значительному увеличению объема продаж и улучшению общего качества обслуживания. Расширение клиентской базы и повышение удобства для пользователей будут способствовать росту количества заказов и улучшению финансовых показателей заведений общественного питания.

Предполагаемые результаты:

1. Рост продаж

2. Увеличение узнаваемости бренда

3. Снижение затрат на управление меню и инвентаризацию

4. Улучшение обслуживания клиентов

Предполагаемый эффект от внедрения:

1. Изменение структуры бизнес-процессов

2. Повышение конкурентоспособности

3. Необходимость постоянного обновления и развития функционала модуля

## 2.2 Требования к функциям, выполняемым системой

Исходя из анализа предметной области, функциональные задачи модуля «Учёт ассортимента блюд для общепита» описаны следующим образом:

Функционал:

1. Обработка данных: Сбор, хранение и обработка информации о блюдах, их ингредиентах, ценах и наличии на складе.

2. Анализ данных: Выявление тенденций и закономерностей на основе заказов и предпочтений клиентов.

3. Управление пользователями: Регистрация и авторизация пользователей, включая сотрудников и администраторов.

4. Интерфейс пользователя: Разработка интуитивно понятного интерфейса для удобной работы с меню и ассортиментом блюд.

5. Мониторинг и безопасность: Контроль состояния системы, защита данных о пользователях и блюдах.

Задачи:

1. Оптимизация процессов: Автоматизация задач, связанных с обновлением ассортимента, расчетом себестоимости и управлением запасами.

2. Поддержка и обновление: Обеспечение технической поддержки для пользователей и регулярное обновление функционала модуля.

3. Сбор обратной связи: Получение отзывов от пользователей о работе модуля для постоянного улучшения его функционала и удобства.

Таблица 1 – Объекты предметной области

|  |  |  |
| --- | --- | --- |
| № | Наименование объекта | Краткое описание |
| 1. | Пользователь | Человек, который имеет определенное право доступа для работы с системой. |
| 2. | Блюдо | Система хранит данные о выполненных действиях пользователя. |
| 3. | Заказ | Окно, в котором представлена информация о конкретном блюда, доступном для покупки. |

После описания объектов предметной области приведем отбор атрибутов для каждого объекта.

Сущность «Пользователь» состоит 4 атрибутов. В ней описаны логин пользователя, пароль, роль и идентификатор.

Таблица 2 – атрибуты сущности «Пользователь»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип | Описание |
| id | INTEGER PRIMARY KEY AUTOINCREMENT | Уникальный идентификатор пользователя |
| login | TEXT UNIQUE | Логин пользователя |
| password | TEXT | Пароль пользователя |
| role | TEXT | Роль пользователя |

Сущность «Блюдо» состоит из 5 атрибутов. В ней описаны название, ингредиенты, цена, количество и идентификатор.

Таблица 3 – атрибуты сущности «Блюдо»

| Тип | Описание |
| --- | --- |
| INTEGER PRIMARY KEY AUTOINCREMENT | Уникальный идентификатор пользователя |
| TEXT | Название блюд |
| TEXT | Состав ингредиентов |
| REAL | Цены блюд |
| INTEGER | Количество блюд |

Сущность «Заказы» состоит из 6 атрибутов. В ней описаны логин пользователя, название блюдо, количество заказа, общая цена, статус и идентификатор.

Таблица 3 – атрибуты сущности «Заказы»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип | Описание |
| id | INTEGER PRIMARY KEY AUTOINCREMENT | Уникальный идентификатор пользователя |
| User\_login | TEXT | Логин заказчика |
| status | TEXT | Статус заказа |
| dish\_id | INTEGER | Блюдо |
| quantity | INTEGER | Количество блюд |
| Total\_price | REAL | Общая цена |

Установим связь 1 к 1 и покажем в “Рисунок 1”

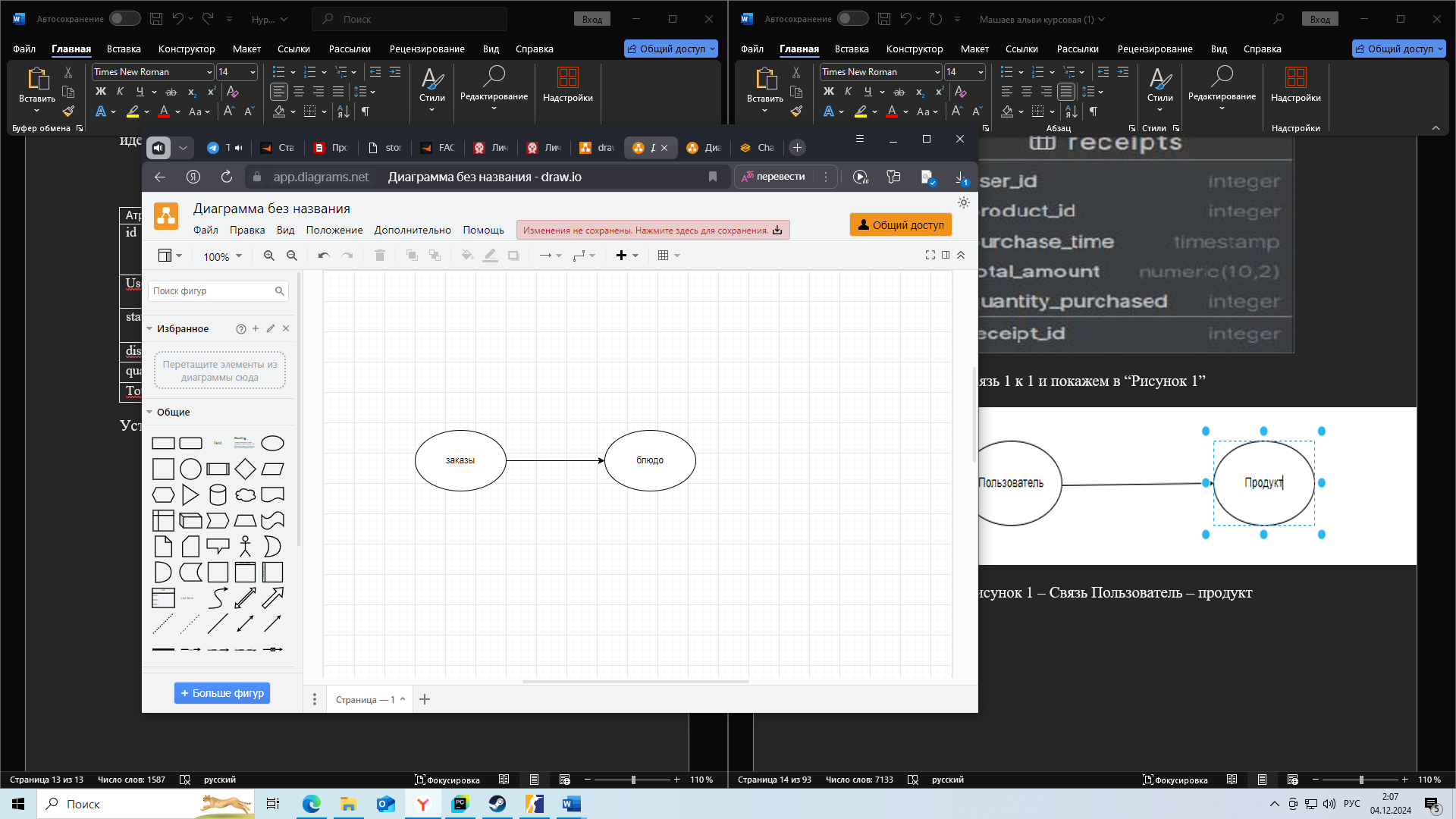


Рисунок 1 – Связь Заказы – Блюдо

Установим связь 1 ко многим и представим в “Рисунок 2”

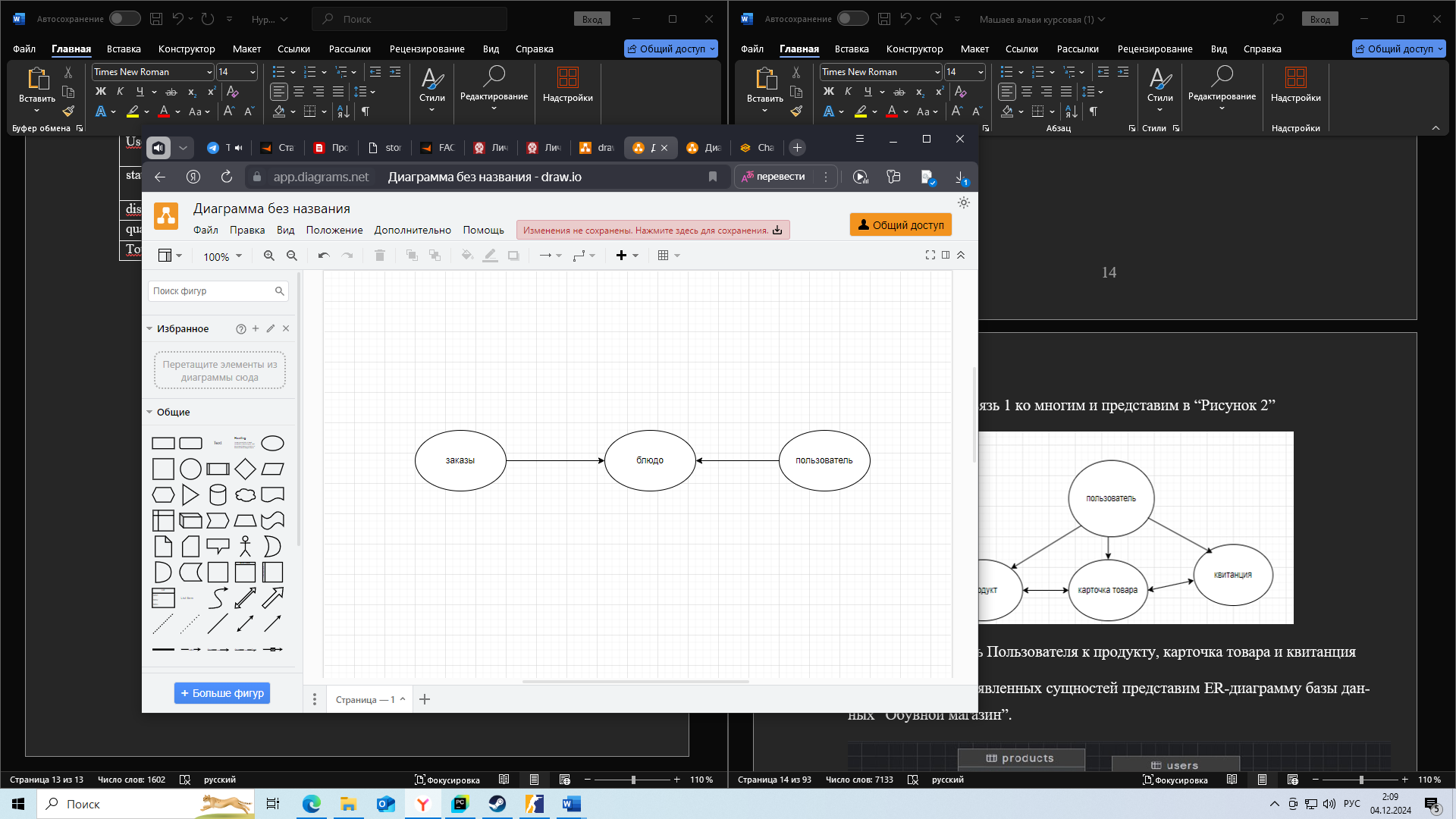


Рисунок 2 – Связь Заказа и Пользователя к Блюдо

На основе выявленных сущностей представим ER-диаграмму база данных “Учёт ассортимента блюд для общепита”

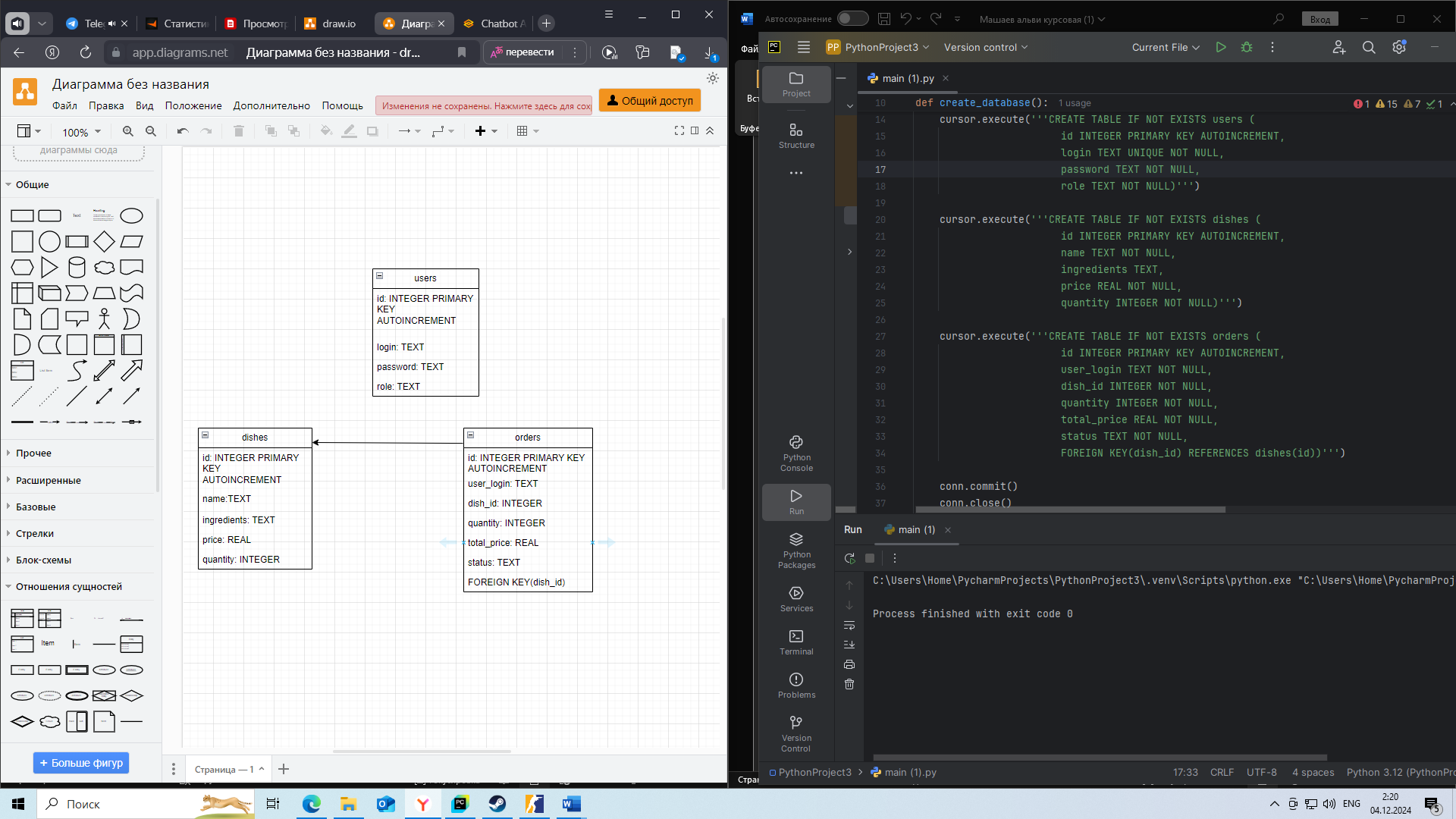


Рисунок 3 – ER-Диаграмма база данных «Учёт ассортимента блюд для общепита»

Для создания диаграммы прецедентности для модуля «Учёт ассортимента блюд для общепита» можно выделить несколько ключевых элементов:

1. Актеры:

- Пользователь (покупатель)

-Администратор

1. Прецеденты:

- Регистрация пользователя

- Добавление нового блюда

- Удаление блюда из ассортимента

- Просмотр заказов

- Создания отчета о продажах

1. Связи между актерами и прецедентами:

- Пользователь может заказывать блюда, и смотреть свои заказы.

- Администратор редактирует информации о блюде и удаляет те блюда, которых нет в наличии.

В “Приложение 1” показана диаграмма прецедентности, которая визуально отображает способы использования системы администратором и пользователем.

## 2.3 План тестирования

Тестирование программного модуля «Учёт ассортимента блюд для общепита» охватывает как функциональные, так и нефункциональные аспекты, что обеспечивает эффективную работу системы. Цель функционального тестирования заключается в том, чтобы убедиться, что все функции работают корректно и соответствуют требованиям заказчика, обеспечивая тем самым надёжность и удобство работы с системой. Цель нефункционального тестирования — убедиться, что модуль корректно функционирует в различных условиях и соответствует установленным стандартам качества.

Задачи тестирования:

1. Функциональное
2. Производительность/стабильность
3. Удобство/графический интерфейс
4. Обеспечение безопасности данных
5. Совместимости с различным ПО

Более подробно с план-тестом в виде таблицы, можно ознакомиться в “Приложение 2”.

Таким образом, тестирование модуля системы «Учёт ассортимента блюд для общепита» охватывает широкий спектр аспектов, направленных на создание надёжного, безопасного и удобного для пользователей приложения. Основная цель тестирования — обеспечить высокое качество работы модуля и максимально удовлетворить потребности пользователей, таких как повара, менеджеры и администраторы, обеспечивая эффективное управление ассортиментом блюд.**3. Реализация проекта системы**

## 3.1 Описание среды разработки (Ide PyCharm)

PyCharm – это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains. PyCharm предлагает широкий спектр функциональности для удобной работы с кодом.

Основные характеристики:

1. Интуитивно понятный интерфейс.
2. Подсветка синтаксиса
3. Поддержка различных фреймворков
4. Интеграция с базами данных

### 3.1.1 Выбор и описание программных инструментов.

Python – это высокоуровневый язык программирования, который используется для разработки приложений, анализа данных и многого другого. Он известен своей простотой, читаемостью кода и хорошими библиотеками. Python подходит для различных приложений, от веб-разработки до анализа данных. [11], [12], [13]

SQLite3 — это легковесная, встроенная реляционная система управления базами данных, которая использует язык SQL для работы с данными. Она разработана для обеспечения простоты, быстродействия и надежности. [15]

PyQt6 — это набор привязок Python для фреймворка Qt 6, который используется для разработки графических интерфейсов пользователя (GUI). С его помощью разработчики могут создавать кроссплатформенные приложения с богатым и интерактивным интерфейсом. [14]

### 3.1.2 Обоснование выбора инструментарий разработки

1. Python:

* Простота изучения
* Широкий спектр применения
* Большое сообщество

1. SQLite3

* Отсутствие настройки
* Подходит для приложений с низкой нагрузкой
* Поддержка SQL
* Лёгкость в использовании

1. PyQt6

* Популярность и сообщество
* Богатый функционал
* Поддержка современной разработки
* Удобство и простота использования

Таким образом выбор инструментов Python, SQLite3 и PyQt6 для разработки программного модуля «Учёта ассортимента блюд для общепита» является обоснованным благодаря их простоте. Эти инструменты позволяют создавать качественные приложения с минимальными затратами времени.

## 3.2 Разработка программного модуля

Создание программного продукта для обувного магазина включает несколько ключевых аспектов:

1. Анализ требований
2. Проектирование интерфейса
3. База данных
4. Функциональность
5. Безопасность
6. Тестирование
7. Запуск и поддержка

### 3.2.1 Реализация пользовательского интерфейса программы

Для обеспечения эффективного взаимодействия пользователя с программным модулем «Учёт ассортимента блюд для общепита» был спроектирован и реализован понятный пользовательский интерфейс

Спроектированный пользовательский интерфейс в среде PyCharm под управлением языка Python выглядит следующим образом:

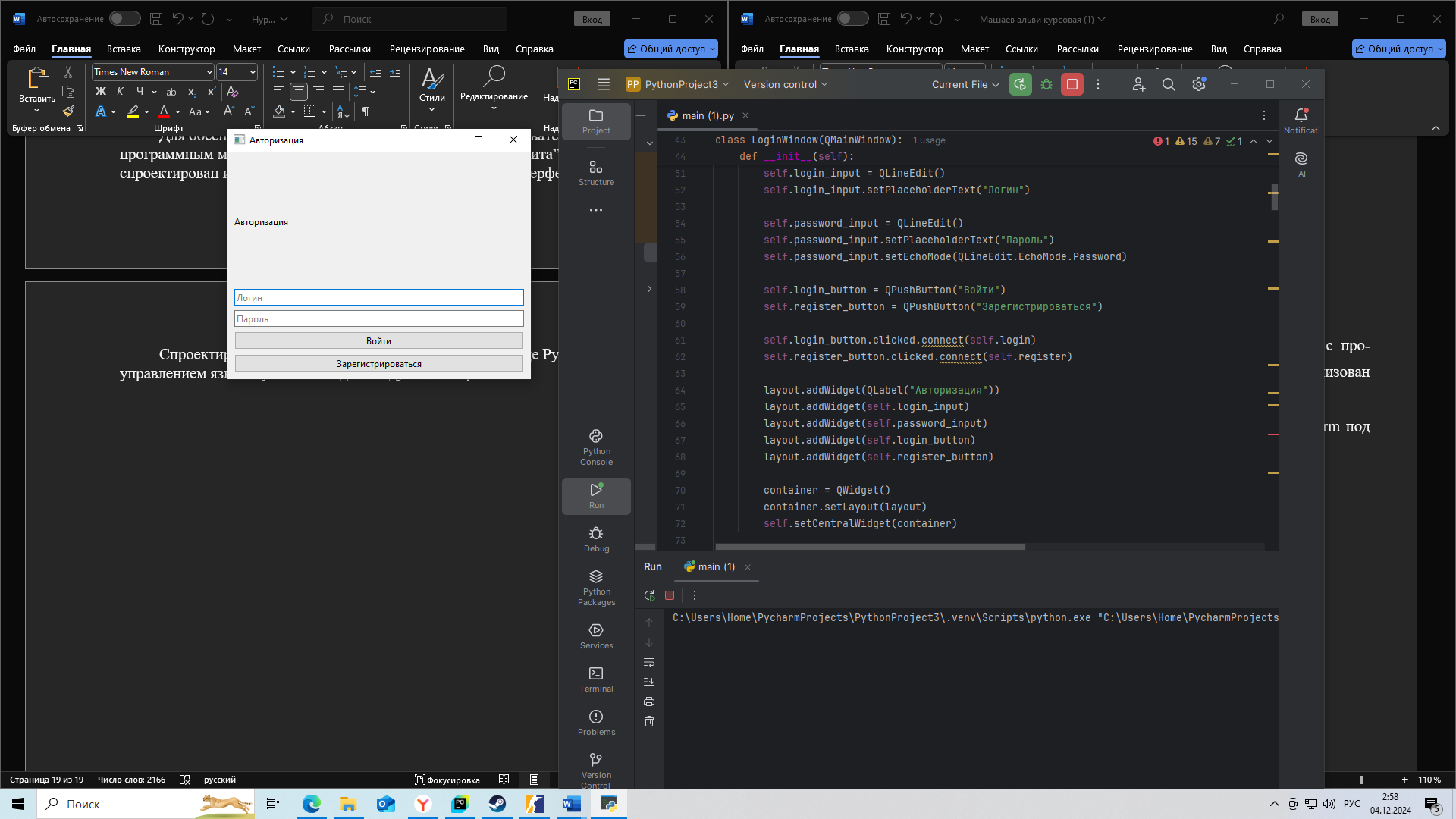


Рисунок 4 – Окно авторизации

- В верхней части окна расположена полоса заголовка с названием “Авторизация” и стандартными кнопками управления окном: свернуть, развернуть и закрыть.

- В нижней части окна расположена рабочая область, которая содержит поле для ввода логина и пароля.

- Внизу окна находится еще две кнопки действий, такие как «Войти» и «Зарегистрироваться», которые предназначены для авторизации пользователя если он не был зарегистрирован и для входа пользователя в систему если он до этого был зарегистрирован.

Так же в системе есть второе пользовательское окно. Окно можно увидеть на «Рисунок 5». Подробнее с ними можно ознакомиться в «Приложение 3».

- В верхней части окна находится меню с названием программы и стандартные кнопки управления окном (свернуть, развернуть, закрыть).

- Под этим находится кнопка с названием выберите блюдо, под ней строчка чтоб указывалась количество заказов.

- В центральной части окна можно оформить заказ и посмотреть все свои заказы

### 3.2.2. Описание кодом функциональных узлов модуля

Модуль «Учёт ассортимента блюд для общепита» состоит из несколько ключевых функциональных узлов, обеспечивающих реализацию взаимодействия пользователей с системой. Представленный код иллюстрирует основные компоненты. Подробнее можно ознакомиться в «Приложение 3».

Функциональный узел: Авторизация пользователей

Этот узел позволяет пользователям входить в систему или регистрироваться. Реализован в «Листинг 1».

def register(self):  
 login = self.login\_input.text()  
 password = self.password\_input.text()  
  
 if not login or not password:  
 QMessageBox.warning(self, "Ошибка", "Заполните все поля!")  
 return  
  
 # Определяем роль: администратор, если пароль 12345, иначе пользователь  
 role = "admin" if password == "12345" else "user"  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 try:  
 cursor.execute("INSERT INTO users (login, password, role) VALUES (?, ?, ?)", (login, password, role))  
 conn.commit()  
 QMessageBox.information(self, "Успех", "Регистрация прошла успешно!")  
 except sqlite3.IntegrityError:  
 QMessageBox.warning(self, "Ошибка", "Такой логин уже существует!")  
 conn.close()

Листинг 1 – Реализация авторизации пользователя

Функциональный узел: Добавление нового блюда

Админ может добавлять блюда. Реализовано в «Листинг 2».

def save\_dish(self):  
 name = self.name\_input.text()  
 ingredients = self.ingredients\_input.text()  
 price = self.price\_input.value()  
 quantity = self.quantity\_input.value()  
  
 if not name or not ingredients or price <= 0 or quantity < 0:  
 QMessageBox.warning(self, "Ошибка", "Пожалуйста, заполните все поля корректно!")  
 return  
  
 conn sqlite3.connect('restaurant.db')  
 cursor conn.cursor()  
  
 if self.is\_edit:  
 cursor.execute('''UPDATE dishes SET name=?, ingredients=?, price=?, quantity=? WHERE id=?''',  
 (name, ingredients, price, quantity, self.dish\_id))  
 QMessageBox.information(self, "Успех", "Блюдо обновлено!")  
 else:  
 cursor.execute('''INSERT INTO dishes (name, ingredients, price, quantity) VALUES (?, ?, ?, ?)''',  
 (name, ingredients, price, quantity))  
 QMessageBox.information(self, "Успех", "Блюдо добавлено!")  
  
 conn.commit()  
 conn.close()  
 self.close()  
 self.parent().load\_dishes()

Листинг 2 – Реализация управления блюдами

Функциональный узел: Удаление блюда из ассортимента.

Админ может удалять блюда из ассортимента. Реализован в «Листинг 3».

def delete\_dish(self):  
 *"""Удаляет выбранное блюдо."""* current\_row = self.table.currentRow()  
 if current\_row < 0:  
 QMessageBox.warning(self, "Ошибка", "Выберите блюдо для удаления!")  
 return  
  
 dish\_id = self.table.item(current\_row, 0).text()  
  
 reply = QMessageBox.question(self, "Удалить блюдо",  
 f"Вы уверены, что хотите удалить блюдо с ID {dish\_id}?",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)  
  
 if reply == QMessageBox.StandardButton.Yes:  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("DELETE FROM dishes WHERE id=?", (dish\_id,))  
 conn.commit()  
 conn.close()  
  
 QMessageBox.information(self, "Успех", "Блюдо удалено!")  
 self.load\_dishes()

Листинг 3 – Реализация удаления блюд из ассортимента

Функциональный узел: Просмотр заказов

Пользователь может смотреть свои заказы. Реализован в «Листинг 4».

def load\_orders(self):  
 *"""Загружает список заказов из базы данных в таблицу."""* conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 # Исправляем запрос с добавлением JOIN с таблицей dishes  
 cursor.execute("""  
 SELECT o.id, o.user\_login, d.name AS dish\_name, o.quantity, o.total\_price, o.status   
 FROM orders o  
 JOIN dishes d ON o.dish\_id = d.id  
 """)  
  
 orders = cursor.fetchall()  
 conn.close()  
  
 self.table.setRowCount(len(orders))  
 self.table.setColumnCount(6)  
 self.table.setHorizontalHeaderLabels(  
 ["ID", "Логин пользователя", "Блюдо", "Количество", "Итоговая цена", "Статус"])  
 self.table.setColumnWidth(1, 200)  
 self.table.setColumnWidth(2, 150)  
 self.table.setColumnWidth(3, 100)  
 self.table.setColumnWidth(4, 150)  
 self.table.setColumnWidth(5, 100)  
  
 for row, order in enumerate(orders):  
 for col, value in enumerate(order[:-1]): # Все кроме статуса  
 self.table.setItem(row, col, QTableWidgetItem(str(value)))  
  
 # Добавляем ComboBox для статуса в последнюю колонку  
 status\_combo = QComboBox()  
 status\_combo.addItem("Готовится")  
 status\_combo.addItem("Готово")  
 status\_combo.setCurrentText(order[5]) # Устанавливаем текущий статус  
  
 status\_combo.currentTextChanged.connect(lambda \_, row=row: self.on\_status\_changed(row))  
 self.table.setCellWidget(row, 5, status\_combo)

Листинг 4 – Реализация просмотр заказов

Функциональный узел: Создания отчета о продажах

Админ может создавать отчеты о продажах. Реализован в «Листинг 5».

def generate\_report(self):  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 # Запрос для получения названия блюда, общего количества проданных блюд и дохода с каждого блюда  
 cursor.execute('''SELECT dishes.name, SUM(orders.quantity) AS total\_quantity, SUM(orders.total\_price) AS total\_revenue  
 FROM orders   
 JOIN dishes ON orders.dish\_id = dishes.id   
 GROUP BY dishes.name''')  
 results = cursor.fetchall()  
 conn.close()  
  
 # Формируем текст отчета  
 report = "Отчет о продажах:\n"  
 total\_revenue = 0 # Общий доход  
 for row in results:  
 dish\_name = row[0]  
 total\_quantity = row[1]  
 revenue = row[2]  
 total\_revenue += revenue  
  
 report += f"{dish\_name}: {total\_quantity} продано, доход: {revenue:.2f} руб.\n"  
  
 # Добавляем общий доход в конец отчета  
 report += f"\nОбщий доход: {total\_revenue:.2f} руб."  
  
 # Отображаем отчет в диалоговом окне  
 QMessageBox.information(self, "Отчет", report)

Листинг 5 – Реализация создания отчета о продажах

Код выше описывает реализацию основных узлов системы. Каждый узел разработан с учётом безопасности и удобства взаимодействия с пользователем.

### 3.2.3 Результаты работы и тестирования

Для подтверждения разработанного модуля были протестированы основные функциональные узлы системы.

Экран 1: Вход администратора

* На этом экране администратор успешно авторизуется в системе.

Результат тестирования:Авторизация успешно выполнена, интерфейс администрация отображается без задержек.

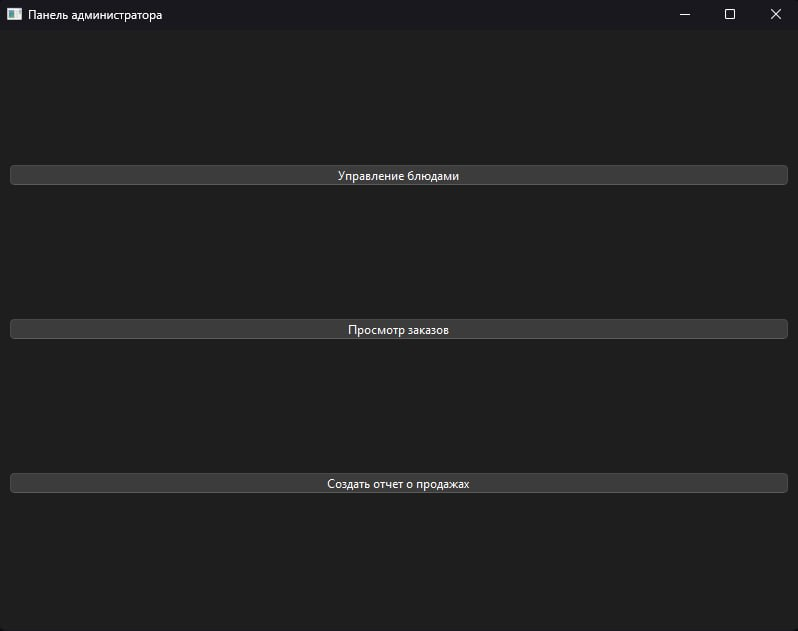


Рисунок 7 – Авторизация «Админ» панели

Экран 2: Добавление блюд администратором

На данном экране отображаются поля для ввода данных о новом блюде.

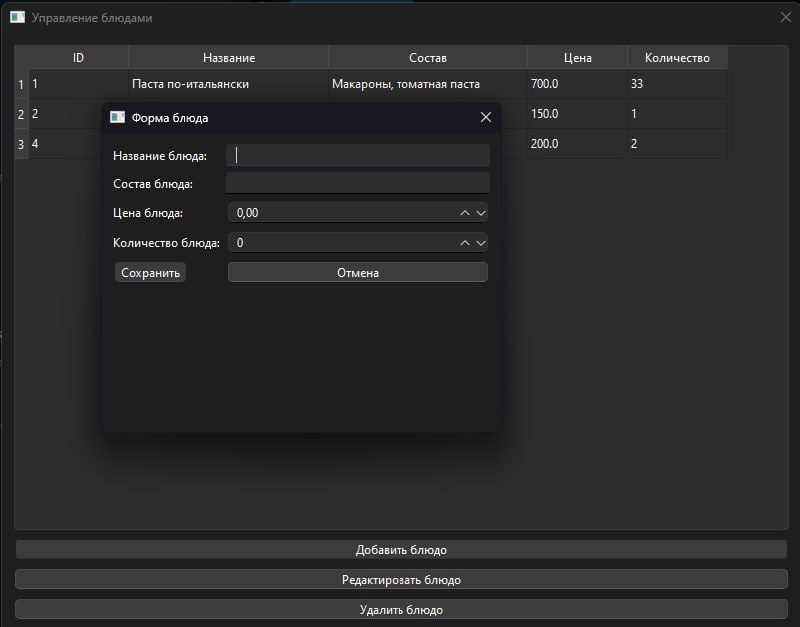


Рисунок 8 – Добавление блюд

Результат тестирования: Блюдо добавлено и отображается у пользователя.

Экран 4: Заказ блюд

После выбора блюд и нажатия “Оформить заказ” пользователь видит в своих заказах что заказ готовится.

Результат тестирования: Информация об историях заказов обновляется в базе данных.

Результатом проекта является программный модуль, который выложен на gethub. [16]

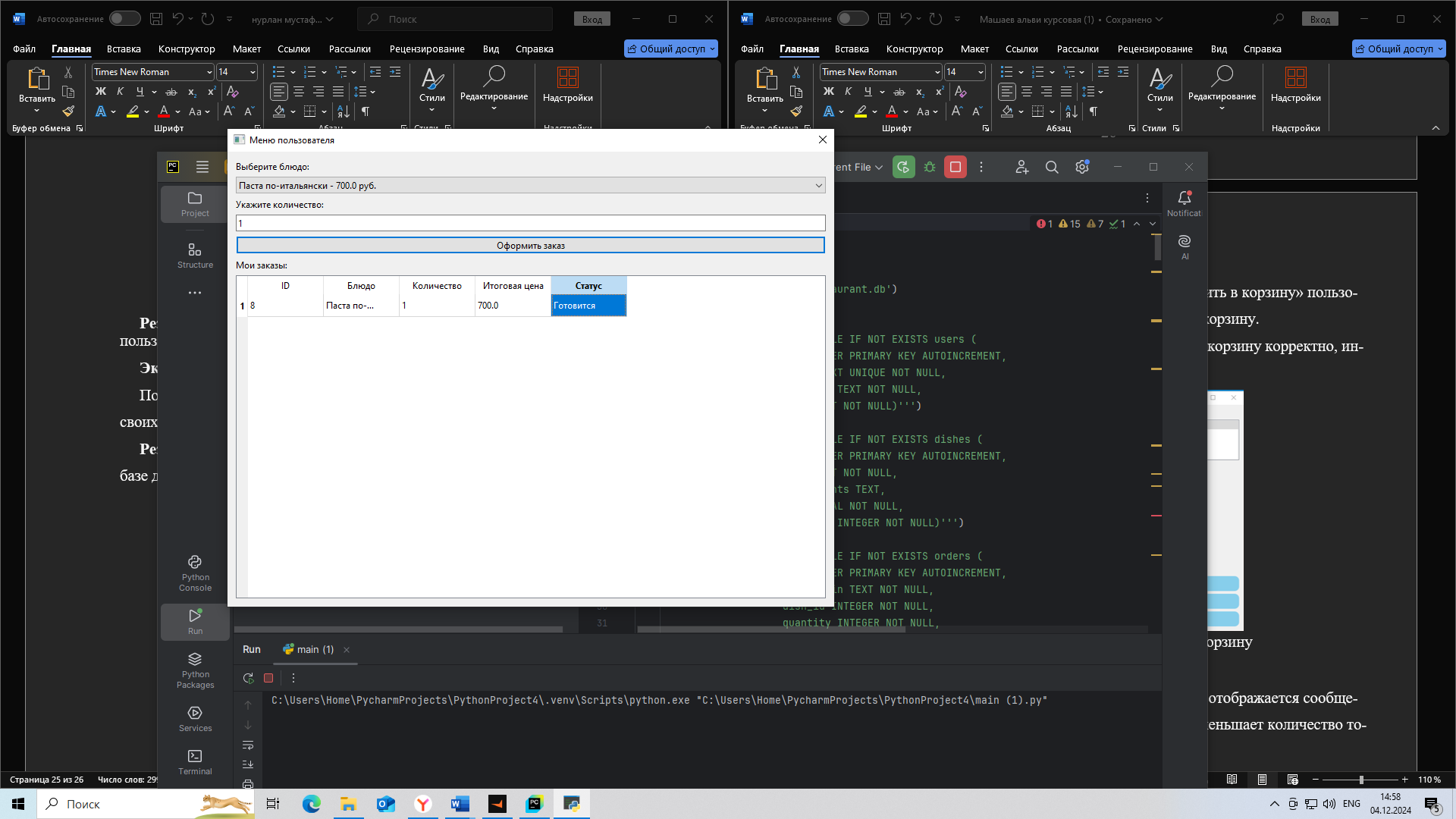


Рисунок 9 - Заказ блюд

# **ЗАКЛЮЧЕНИЕ**

В ходе разработки программного модуля "Учёт ассортимента блюд для общепита" была решена важная задача оптимизации процессов управления меню и учёта блюд в заведениях общественного питания. Современные условия работы ресторанного и кофейного бизнеса требуют от менеджеров и поваров быстрой адаптации к изменениям в потребительских предпочтениях, что стало возможным благодаря внедрению данного модуля.

Разработанный модуль позволяет автоматизировать процессы учета, что существенно сокращает временные затраты на внесение данных, мониторинг ассортимента и составление отчетов. Интуитивно понятный интерфейс и возможность быстрого поиска информации о блюдах способствуют эффективной работе сотрудников.

Важно отметить, что модуль обеспечивает гибкость в управлении ассортиментом: пользователи могут легко добавлять новые блюда, изменять характеристики существующих и удалять неактуальные позиции. Также была реализована функция анализа продаж, что позволяет принимать более обоснованные решения о меню, основываясь на реальных данных о предпочтениях клиентов.

Внедрение данного программного решения не только улучшает качество обслуживания, но и способствует увеличению продаж за счет более эффективного предложения блюд. С ростом конкуренции в сфере общепита, подобные модули становятся необходимостью для поддержания конкурентоспособности и повышения лояльности клиентов.

Таким образом, разработка модуля "Учёт ассортимента блюд для общепита" является значимым шагом на пути к модернизации процессов управления в сфере общественного питания, открывающим новые возможности как для владельцев заведений, так и для их клиентов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Нормативно-правовые источники

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

2. ГОСТ 7.0.100-2018 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

3. ГОСТ 7.9-95 Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования.

4. ГОСТ 7.11-2004 (ИСО 832:1994) Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов и словосочетаний на иностранных европейских языках.

5. ГОСТ Р 7.0.12-2011 Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись сокращение слов и словосочетаний на русском языке. Общие требования и правила.

6. ГОСТ 7.79-2000 (ИСО 9-95) Система стандартов по информации, библиотечному и издательскому делу. Правила транслитерации кирилловского письма латинским алфавитом.

7. ГОСТ 7.80-2000 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления.

8. ГОСТ 7.90-2007 Система стандартов по информации, библиотечному и издательскому делу. Универсальная десятичная классификация. Структура, правила ведения и индексирования.

9. ГОСТ 8.417-2002 Государственная система обеспечения единства измерений. Единицы величин.

10. ГОСТ 9327-60 Бумага и изделия из бумаги. Потребительские форматы.

Учебники, учебные пособия, статьи

11. Введение в разработку приложений на Python / Пол Бэрри. Издательство «ЭКСМО», 2019. 615 с.

12. “Python 3. Эффективное программирование” / Бретт Слэткин. Москва, 2019 г., Издательство “Питер”. 624 с.

Интернет-источники

13. Python 3.10 Documentation - доступно на официальном сайте Python: https://docs.python.org/3.10/ (дата обращения: 31.11.2024).

14.Руководство по разработке на PyQt6 – https://www.riverbankcomputing.com/software/pyqt/intro (дата обращения: 25.11.2024).

15. SQLite Documentation - доступно на официальном сайте SQLite: https://www.sqlite.org/docs.html (дата обращения: 25.11.2024).

Монография

16. Курсовой проект по теме «Разработка программного модуля

«Учёт ассортимента блюд для общепита»». https://github.com/Nurlan1488/Work

# **ПРИЛОЖЕНИЕ 1**

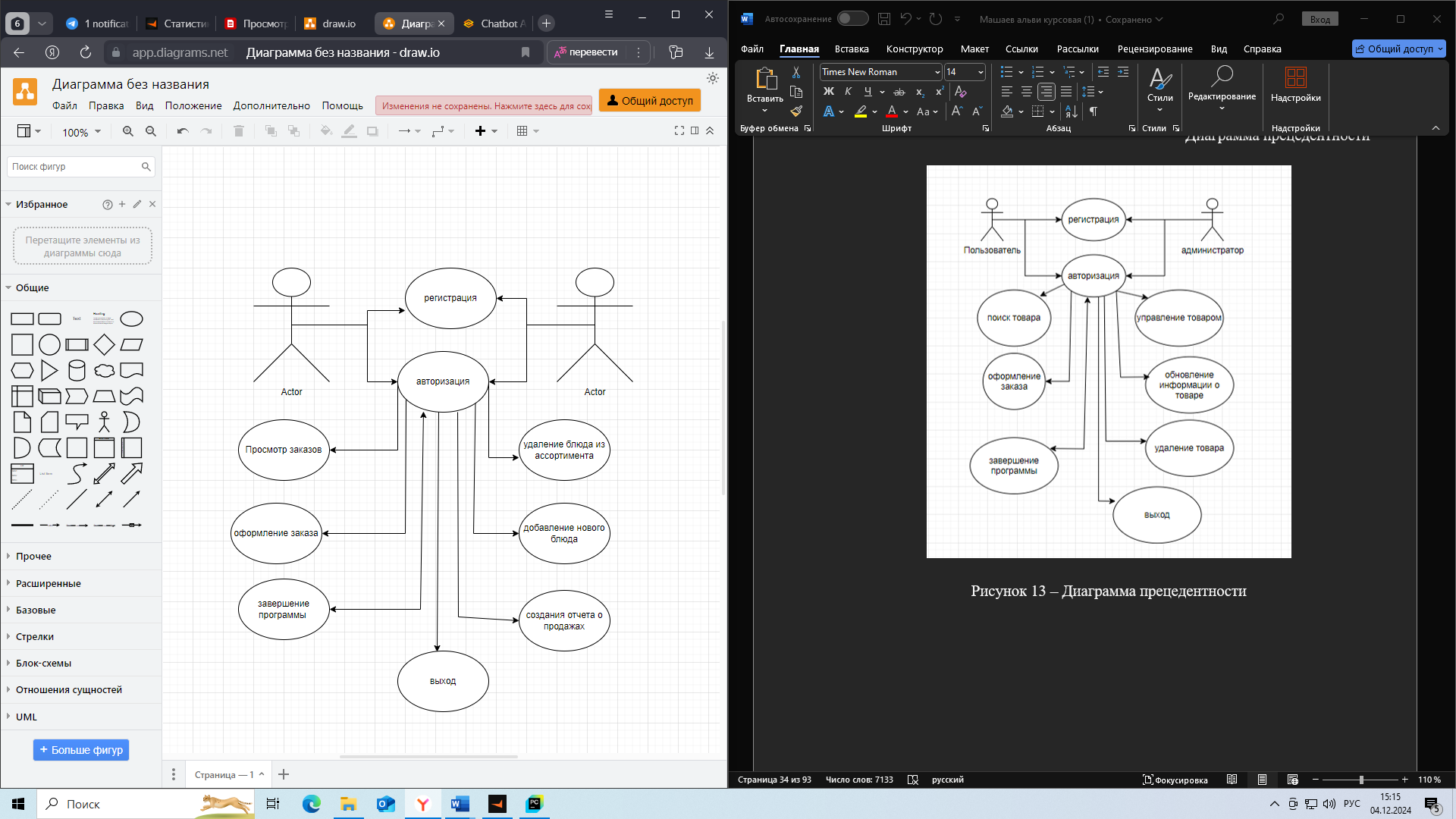


Рисунок 10 – Диаграмма прецедентности

# **ПРИЛОЖЕНИЕ 2**

Таблица 4 - План-тест

| **№** | **Наименование функциональности** | **Наименование поля** | **Тестовый набор** | **Результат (должно получиться)** | **Результат тестирования** |
| --- | --- | --- | --- | --- | --- |
| 1 | Авторизация пользователя | Логин, пароль | Ввести корректный логин и пароль, нажать "Войти" | Пользователь успешно авторизован | Пользователь был авторизован в системе |
|  |  |  | Проверить отображение приветственного сообщения | Сообщение: "Добро пожаловать, [имя пользователя]" | Сообщение корректно отображено |
|  |  |  | Проверить доступ к пользовательской панели | Переход на пользовательскую панель | Доступ к панели предоставлен |
|  |  |  | Проверить запись успешной авторизации в логи | Лог содержит запись о корректной авторизации | Запись об авторизации была добавлена |
|  |  |  | Проверить состояние сессии | Сессия активирована и привязана к текущему пользователю | Сессия успешно активирована |
| 2 | Авторизация пользователя | Логин, пароль | Ввести некорректный логин или пароль, нажать "Войти" | Сообщение: "Неверный логин или пароль" | Сообщение отображено корректно |
|  |  |  | |  | | --- | | Проверить отсутствие перехода на пользовательскую панель | | Остаётся на экране авторизации | Перехода на панель не произошло |
|  |  |  | Проверить состояние сессии | Сессия не активируется | Сессия осталась неактивной |
|  |  |  | Попробовать повторную авторизацию после ввода некорректных данных | Сообщение повторно отображается при каждом вводе неверных данных | Повторные сообщения корректны |
|  |  |  | Ввести данные, превышающие допустимую длину логина или пароля | Сообщение: "Логин/пароль превышает допустимую длину" | Сообщение корректно отображено |
| 3 | Авторизация пользователя | Логин, пароль | Оставить поля логина и пароля пустыми, нажать "Войти" | Сообщение: "Заполните все поля" | Сообщение отображено корректно |
|  |  |  | Проверить, остаётся ли пользователь на экране авторизации | Остаётся на текущем экране | Перехода на другой экран не произошло |
|  |  |  | Ввести только логин, оставить поле пароля пустым | Сообщение: "Заполните поле пароль" | Сообщение корректно отображено |
|  |  |  | Ввести только пароль, оставить поле логина пустым | Сообщение: "Заполните поле логин" | Сообщение корректно отображено |
|  |  |  | Нажать "Войти" без ввода данных повторно | Повторное сообщение: "Заполните все поля" | Повторное сообщение корректно отображено |
|  |  |  | Проверить запись в логах при отсутствии данных | Лог не содержит записи о попытке авторизации | Лог записей корректный |
| 4 | Управление блюдами | Название, цена, состав, количество. | Ввести корректные данные для названия, цены, количества и состава | Блюд успешно добавлена в систему | Блюдо добавлена, данные отображены |
|  |  |  | Проверить отображение новых блюд в каталоге | Новый ассортимент отображается в каталоге | Товар отображается корректно |
|  |  |  | Выбрать существующие блюда из списка | Товар выделен | Товар успешно выделен |
|  |  |  | Нажать кнопку “Удалить” | Сообщение: "Вы уверены, что хотите удалить товар?" | Сообщение отображено корректно |
|  |  |  | Подтвердить удаление | Товар удалён из системы | Товар успешно удалён |
|  |  |  | Проверить отображение списка после удаления ассортимента | Удалённое блюдо отсутствует в списке | Товар больше не отображается |
| 5 | Работа с заказом | ассортимент | Выбрать блюдо из предоставленных администратором | Товар выделен | Товар успешно выделен |
|  |  |  | Нажать кнопку заказать | Сообщение: “блюдо готовится” | Сообщение отображено корректно |
|  |  |  | Посмотреть свои заказы | Все заказы успешно показываются | Заказы показываются |
|  |  |  |  |  |  |
| 5 | Создания отчета о продажах | Отчеты продаж | Нажать в окне на кнопку “создать отчет о продажах” | Сообщение “отчет о продажах” с общим доходом | Отчет успешно выполнен |

# **ПРИЛОЖЕНИЕ 3**

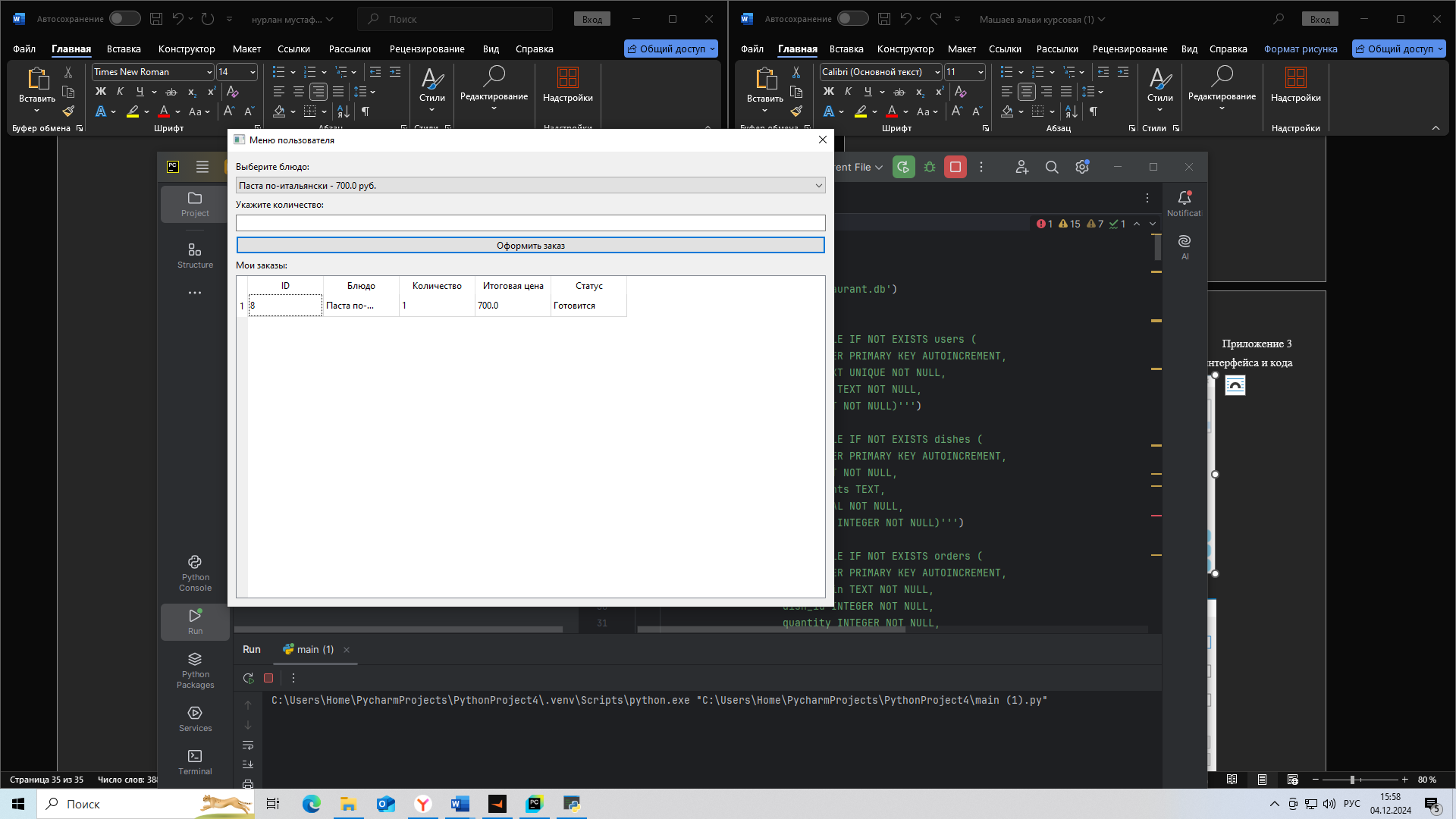


Рисунок 5 – Главное меню

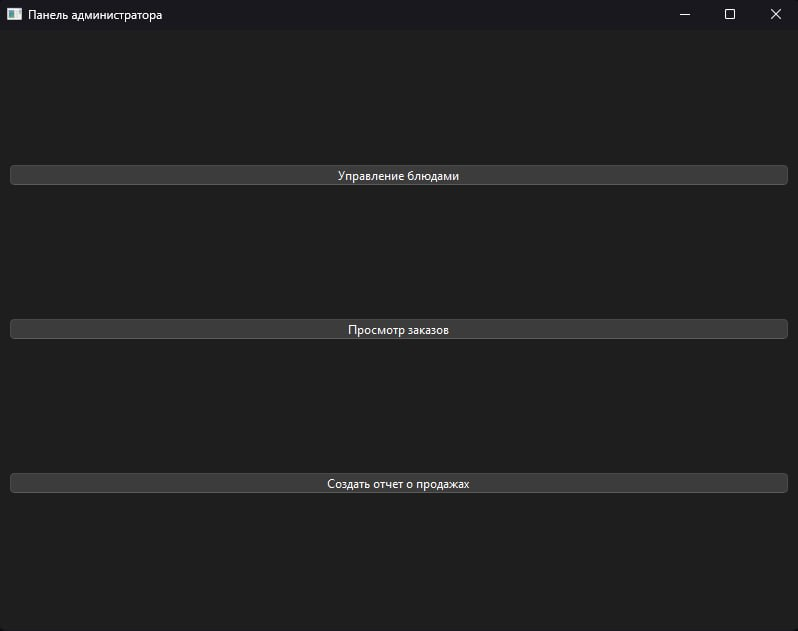


Рисунок 6 – Панель админа

# **ПРИЛОЖЕНИЕ 4**

main.py

import sys  
import sqlite3  
from PyQt6.QtWidgets import (  
 QApplication, QMainWindow, QVBoxLayout, QPushButton, QWidget, QLineEdit, QLabel,  
 QMessageBox, QDialog, QTableWidget, QTableWidgetItem, QSpinBox, QComboBox, QInputDialog, QFormLayout, QDoubleSpinBox  
)  
from PyQt6.QtCore import Qt  
  
# --- Создание базы данных ---  
def create\_database():  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 cursor.execute('''CREATE TABLE IF NOT EXISTS users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 login TEXT UNIQUE NOT NULL,  
 password TEXT NOT NULL,  
 role TEXT NOT NULL)''')  
  
 cursor.execute('''CREATE TABLE IF NOT EXISTS dishes (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 ingredients TEXT,  
 price REAL NOT NULL,  
 quantity INTEGER NOT NULL)''')  
  
 cursor.execute('''CREATE TABLE IF NOT EXISTS orders (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 user\_login TEXT NOT NULL,  
 dish\_id INTEGER NOT NULL,  
 quantity INTEGER NOT NULL,  
 total\_price REAL NOT NULL,  
 status TEXT NOT NULL,  
 FOREIGN KEY(dish\_id) REFERENCES dishes(id))''')  
  
 conn.commit()  
 conn.close()  
  
create\_database()  
  
  
# --- Окно авторизации ---  
class LoginWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Авторизация")  
 self.setGeometry(300, 200, 400, 300)  
  
 layout = QVBoxLayout()  
  
 self.login\_input = QLineEdit()  
 self.login\_input.setPlaceholderText("Логин")  
  
 self.password\_input = QLineEdit()  
 self.password\_input.setPlaceholderText("Пароль")  
 self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)  
  
 self.login\_button = QPushButton("Войти")  
 self.register\_button = QPushButton("Зарегистрироваться")  
  
 self.login\_button.clicked.connect(self.login)  
 self.register\_button.clicked.connect(self.register)  
  
 layout.addWidget(QLabel("Авторизация"))  
 layout.addWidget(self.login\_input)  
 layout.addWidget(self.password\_input)  
 layout.addWidget(self.login\_button)  
 layout.addWidget(self.register\_button)  
  
 container = QWidget()  
 container.setLayout(layout)  
 self.setCentralWidget(container)  
  
 def login(self):  
 login = self.login\_input.text()  
 password = self.password\_input.text()  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("SELECT role FROM users WHERE login = ? AND password = ?", (login, password))  
 result = cursor.fetchone()  
 conn.close()  
  
 if result:  
 role = result[0]  
 if role == "admin":  
 self.admin\_window = AdminWindow()  
 self.admin\_window.show()  
 else:  
 self.user\_window = UserWindow(login)  
 self.user\_window.show()  
 self.close()  
 else:  
 QMessageBox.warning(self, "Ошибка", "Неверный логин или пароль!")  
  
 def register(self):  
 login = self.login\_input.text()  
 password = self.password\_input.text()  
  
 if not login or not password:  
 QMessageBox.warning(self, "Ошибка", "Заполните все поля!")  
 return  
  
 # Определяем роль: администратор, если пароль 12345, иначе пользователь  
 role = "admin" if password == "12345" else "user"  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 try:  
 cursor.execute("INSERT INTO users (login, password, role) VALUES (?, ?, ?)", (login, password, role))  
 conn.commit()  
 QMessageBox.information(self, "Успех", "Регистрация прошла успешно!")  
 except sqlite3.IntegrityError:  
 QMessageBox.warning(self, "Ошибка", "Такой логин уже существует!")  
 conn.close()  
  
  
class AdminWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Панель администратора")  
 self.setGeometry(300, 200, 800, 600)  
  
 layout = QVBoxLayout()  
  
 self.btn\_manage\_dishes = QPushButton("Управление блюдами")  
 self.btn\_manage\_orders = QPushButton("Просмотр заказов")  
 self.btn\_generate\_report = QPushButton("Создать отчет о продажах")  
  
 self.btn\_manage\_dishes.clicked.connect(self.manage\_dishes)  
 self.btn\_manage\_orders.clicked.connect(self.view\_orders)  
 self.btn\_generate\_report.clicked.connect(self.generate\_report)  
  
 layout.addWidget(self.btn\_manage\_dishes)  
 layout.addWidget(self.btn\_manage\_orders)  
 layout.addWidget(self.btn\_generate\_report)  
  
 container = QWidget()  
 container.setLayout(layout)  
 self.setCentralWidget(container)  
  
 def manage\_dishes(self):  
 self.dish\_window = DishManagementWindow()  
 self.dish\_window.show()  
  
 def view\_orders(self):  
 self.order\_window = OrderManagementWindow()  
 self.order\_window.show()  
  
 def generate\_report(self):  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 # Запрос для получения названия блюда, общего количества проданных блюд и дохода с каждого блюда  
 cursor.execute('''SELECT dishes.name, SUM(orders.quantity) AS total\_quantity, SUM(orders.total\_price) AS total\_revenue  
 FROM orders   
 JOIN dishes ON orders.dish\_id = dishes.id   
 GROUP BY dishes.name''')  
 results = cursor.fetchall()  
 conn.close()  
  
 # Формируем текст отчета  
 report = "Отчет о продажах:\n"  
 total\_revenue = 0 # Общий доход  
 for row in results:  
 dish\_name = row[0]  
 total\_quantity = row[1]  
 revenue = row[2]  
 total\_revenue += revenue  
  
 report += f"{dish\_name}: {total\_quantity} продано, доход: {revenue:.2f} руб.\n"  
  
 # Добавляем общий доход в конец отчета  
 report += f"\nОбщий доход: {total\_revenue:.2f} руб."  
  
 # Отображаем отчет в диалоговом окне  
 QMessageBox.information(self, "Отчет", report)  
  
  
# --- Управление блюдами ---  
class DishManagementWindow(QDialog):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Управление блюдами")  
 self.setGeometry(300, 200, 800, 600)  
  
 layout = QVBoxLayout()  
  
 # Таблица для отображения блюд  
 self.table = QTableWidget(self)  
 self.load\_dishes()  
  
 # Кнопки для управления блюдами  
 self.btn\_add = QPushButton("Добавить блюдо")  
 self.btn\_edit = QPushButton("Редактировать блюдо")  
 self.btn\_delete = QPushButton("Удалить блюдо")  
  
 # Привязка кнопок к функциям  
 self.btn\_add.clicked.connect(self.add\_dish)  
 self.btn\_edit.clicked.connect(self.edit\_dish)  
 self.btn\_delete.clicked.connect(self.delete\_dish)  
  
 layout.addWidget(self.table)  
 layout.addWidget(self.btn\_add)  
 layout.addWidget(self.btn\_edit)  
 layout.addWidget(self.btn\_delete)  
  
 self.setLayout(layout)  
  
 def load\_dishes(self):  
 *"""Загружает список блюд из базы данных в таблицу."""* conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("SELECT id, name, ingredients, price, quantity FROM dishes")  
 dishes = cursor.fetchall()  
 conn.close()  
  
 self.table.setRowCount(len(dishes))  
 self.table.setColumnCount(5)  
 self.table.setHorizontalHeaderLabels(["ID", "Название", "Состав", "Цена", "Количество"])  
 self.table.setColumnWidth(1, 200)  
 self.table.setColumnWidth(2, 200)  
 self.table.setColumnWidth(3, 100)  
 self.table.setColumnWidth(4, 100)  
  
 for row, dish in enumerate(dishes):  
 for col, value in enumerate(dish):  
 item = QTableWidgetItem(str(value))  
 item.setFlags(item.flags() & ~Qt.ItemFlag.ItemIsEditable if col == 0 else item.flags())  
 self.table.setItem(row, col, item)  
  
 def add\_dish(self):  
 *"""Добавляет новое блюдо в базу данных."""* self.dish\_form = DishFormWindow(self, is\_edit=False)  
 self.dish\_form.show()  
  
 def edit\_dish(self):  
 *"""Редактирует выбранное блюдо."""* current\_row = self.table.currentRow()  
 if current\_row < 0:  
 QMessageBox.warning(self, "Ошибка", "Выберите блюдо для редактирования!")  
 return  
  
 dish\_id = self.table.item(current\_row, 0).text()  
 dish\_name = self.table.item(current\_row, 1).text()  
 dish\_ingredients = self.table.item(current\_row, 2).text()  
 dish\_price = float(self.table.item(current\_row, 3).text())  
 dish\_quantity = int(self.table.item(current\_row, 4).text())  
  
 self.dish\_form = DishFormWindow(self, is\_edit=True, dish\_id=dish\_id,  
 name=dish\_name, ingredients=dish\_ingredients,  
 price=dish\_price, quantity=dish\_quantity)  
 self.dish\_form.show()  
  
 def delete\_dish(self):  
 *"""Удаляет выбранное блюдо."""* current\_row = self.table.currentRow()  
 if current\_row < 0:  
 QMessageBox.warning(self, "Ошибка", "Выберите блюдо для удаления!")  
 return  
  
 dish\_id = self.table.item(current\_row, 0).text()  
  
 reply = QMessageBox.question(self, "Удалить блюдо",  
 f"Вы уверены, что хотите удалить блюдо с ID {dish\_id}?",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)  
  
 if reply == QMessageBox.StandardButton.Yes:  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("DELETE FROM dishes WHERE id=?", (dish\_id,))  
 conn.commit()  
 conn.close()  
  
 QMessageBox.information(self, "Успех", "Блюдо удалено!")  
 self.load\_dishes()  
  
  
# --- Окно формы для добавления/редактирования блюда ---  
class DishFormWindow(QDialog):  
 def \_\_init\_\_(self, parent, is\_edit=False, dish\_id=None, name="", ingredients="", price=0.0, quantity=0):  
 super().\_\_init\_\_(parent)  
 self.setWindowTitle("Форма блюда")  
 self.setGeometry(400, 300, 400, 300)  
  
 self.is\_edit = is\_edit  
 self.dish\_id = dish\_id  
  
 layout = QFormLayout()  
  
 self.name\_input = QLineEdit(name)  
 self.ingredients\_input = QLineEdit(ingredients)  
  
 # Убираем ограничения для цены  
 self.price\_input = QDoubleSpinBox()  
 self.price\_input.setValue(price)  
 self.price\_input.setMinimum(0) # Цена не ограничена верхним значением  
 self.price\_input.setMaximum(9999999) # Ограничиваем на максимально возможное число для удобства  
  
 # Убираем ограничения для количества  
 self.quantity\_input = QSpinBox()  
 self.quantity\_input.setValue(quantity)  
 self.quantity\_input.setMinimum(0) # Количество не ограничено верхним значением  
 self.quantity\_input.setMaximum(9999999) # Ограничиваем на максимально возможное число для удобства  
  
 self.save\_button = QPushButton("Сохранить")  
 self.cancel\_button = QPushButton("Отмена")  
  
 self.save\_button.clicked.connect(self.save\_dish)  
 self.cancel\_button.clicked.connect(self.close)  
  
 layout.addRow("Название блюда:", self.name\_input)  
 layout.addRow("Состав блюда:", self.ingredients\_input)  
 layout.addRow("Цена блюда:", self.price\_input)  
 layout.addRow("Количество блюда:", self.quantity\_input)  
 layout.addRow(self.save\_button, self.cancel\_button)  
  
 self.setLayout(layout)  
  
 # Устанавливаем начальные значения  
 if self.is\_edit:  
 self.price\_input.setValue(price) # Обновляем цену  
 self.quantity\_input.setValue(quantity) # Обновляем количество  
  
 def save\_dish(self):  
 name = self.name\_input.text()  
 ingredients = self.ingredients\_input.text()  
 price = self.price\_input.value()  
 quantity = self.quantity\_input.value()  
  
 if not name or not ingredients or price <= 0 or quantity < 0:  
 QMessageBox.warning(self, "Ошибка", "Пожалуйста, заполните все поля корректно!")  
 return  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 if self.is\_edit:  
 cursor.execute('''UPDATE dishes SET name=?, ingredients=?, price=?, quantity=? WHERE id=?''',  
 (name, ingredients, price, quantity, self.dish\_id))  
 QMessageBox.information(self, "Успех", "Блюдо обновлено!")  
 else:  
 cursor.execute('''INSERT INTO dishes (name, ingredients, price, quantity) VALUES (?, ?, ?, ?)''',  
 (name, ingredients, price, quantity))  
 QMessageBox.information(self, "Успех", "Блюдо добавлено!")  
  
 conn.commit()  
 conn.close()  
 self.close()  
 self.parent().load\_dishes()  
  
class OrderManagementWindow(QDialog):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Управление заказами")  
 self.setGeometry(300, 200, 800, 600)  
  
 layout = QVBoxLayout()  
  
 # Таблица для отображения заказов  
 self.table = QTableWidget(self)  
 self.load\_orders()  
  
 # Кнопки для управления заказами  
 self.btn\_update\_status = QPushButton("Обновить статус")  
  
 # Привязка кнопки к функции  
 self.btn\_update\_status.clicked.connect(self.update\_status)  
  
 layout.addWidget(self.table)  
 layout.addWidget(self.btn\_update\_status)  
  
 self.setLayout(layout)  
  
 def load\_orders(self):  
 *"""Загружает список заказов из базы данных в таблицу."""* conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 # Исправляем запрос с добавлением JOIN с таблицей dishes  
 cursor.execute("""  
 SELECT o.id, o.user\_login, d.name AS dish\_name, o.quantity, o.total\_price, o.status   
 FROM orders o  
 JOIN dishes d ON o.dish\_id = d.id  
 """)  
  
 orders = cursor.fetchall()  
 conn.close()  
  
 self.table.setRowCount(len(orders))  
 self.table.setColumnCount(6)  
 self.table.setHorizontalHeaderLabels(  
 ["ID", "Логин пользователя", "Блюдо", "Количество", "Итоговая цена", "Статус"])  
 self.table.setColumnWidth(1, 200)  
 self.table.setColumnWidth(2, 150)  
 self.table.setColumnWidth(3, 100)  
 self.table.setColumnWidth(4, 150)  
 self.table.setColumnWidth(5, 100)  
  
 for row, order in enumerate(orders):  
 for col, value in enumerate(order[:-1]): # Все кроме статуса  
 self.table.setItem(row, col, QTableWidgetItem(str(value)))  
  
 # Добавляем ComboBox для статуса в последнюю колонку  
 status\_combo = QComboBox()  
 status\_combo.addItem("Готовится")  
 status\_combo.addItem("Готово")  
 status\_combo.setCurrentText(order[5]) # Устанавливаем текущий статус  
  
 status\_combo.currentTextChanged.connect(lambda \_, row=row: self.on\_status\_changed(row))  
 self.table.setCellWidget(row, 5, status\_combo)  
  
 def update\_status(self):  
 *"""Обновляет статус выбранного заказа в базе данных."""* selected\_row = self.table.currentRow()  
 if selected\_row == -1:  
 QMessageBox.warning(self, "Ошибка", "Выберите заказ для изменения статуса!")  
 return  
  
 order\_id = self.table.item(selected\_row, 0).text()  
 status\_combo = self.table.cellWidget(selected\_row, 5)  
 new\_status = status\_combo.currentText()  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("UPDATE orders SET status=? WHERE id=?", (new\_status, order\_id))  
 conn.commit()  
 conn.close()  
  
 QMessageBox.information(self, "Успех", "Статус заказа обновлен!")  
 self.load\_orders()  
  
 def on\_status\_changed(self, row):  
 *"""Сохраняет новый статус в базе данных при изменении выбора в ComboBox."""* order\_id = self.table.item(row, 0).text()  
 status\_combo = self.table.cellWidget(row, 5)  
 new\_status = status\_combo.currentText()  
  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("UPDATE orders SET status=? WHERE id=?", (new\_status, order\_id))  
 conn.commit()  
 conn.close()  
  
# --- Панель пользователя ---  
class UserWindow(QDialog):  
 def \_\_init\_\_(self, user\_login):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Меню пользователя")  
 self.setGeometry(300, 200, 800, 600)  
  
 self.user\_login = user\_login # Сохраняем логин пользователя  
  
 layout = QVBoxLayout()  
  
 # Секция для выбора блюда  
 self.dish\_combo = QComboBox()  
 self.quantity\_input = QLineEdit()  
 self.order\_button = QPushButton("Оформить заказ")  
 self.orders\_table = QTableWidget(self)  
  
 # Загружаем блюда из базы данных  
 self.load\_dishes()  
  
 # Кнопка для оформления заказа  
 self.order\_button.clicked.connect(self.place\_order)  
  
 # Загружаем список заказов пользователя  
 self.load\_orders()  
  
 layout.addWidget(QLabel("Выберите блюдо:"))  
 layout.addWidget(self.dish\_combo)  
 layout.addWidget(QLabel("Укажите количество:"))  
 layout.addWidget(self.quantity\_input)  
 layout.addWidget(self.order\_button)  
  
 # Таблица для отображения заказов  
 layout.addWidget(QLabel("Мои заказы:"))  
 layout.addWidget(self.orders\_table)  
  
 self.setLayout(layout)  
  
 def load\_dishes(self):  
 *"""Загружает блюда из базы данных в выпадающий список"""* self.dish\_combo.clear()  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("SELECT id, name, price, quantity FROM dishes")  
 dishes = cursor.fetchall()  
 conn.close()  
  
 for dish in dishes:  
 self.dish\_combo.addItem(f"{dish[1]} - {dish[2]} руб.", dish[0])  
  
 def load\_orders(self):  
 *"""Загружает список заказов пользователя из базы данных"""* conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute(  
 "SELECT o.id, d.name AS dish\_name, o.quantity, o.total\_price, o.status FROM orders o JOIN dishes d ON o.dish\_id = d.id WHERE o.user\_login = ?",  
 (self.user\_login,))  
 orders = cursor.fetchall()  
 conn.close()  
  
 self.orders\_table.setRowCount(len(orders))  
 self.orders\_table.setColumnCount(5)  
 self.orders\_table.setHorizontalHeaderLabels(["ID", "Блюдо", "Количество", "Итоговая цена", "Статус"])  
  
 for row, order in enumerate(orders):  
 for col, value in enumerate(order):  
 self.orders\_table.setItem(row, col, QTableWidgetItem(str(value)))  
  
 def place\_order(self):  
 *"""Оформляет заказ пользователя"""* dish\_id = self.dish\_combo.currentData()  
 quantity = self.quantity\_input.text()  
  
 if not quantity.isdigit() or int(quantity) <= 0:  
 QMessageBox.warning(self, "Ошибка", "Количество должно быть положительным числом!")  
 return  
  
 quantity = int(quantity)  
  
 # Проверяем наличие блюда  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
 cursor.execute("SELECT quantity, price FROM dishes WHERE id = ?", (dish\_id,))  
 dish = cursor.fetchone()  
 conn.close()  
  
 available\_quantity, price = dish  
  
 if quantity > available\_quantity:  
 QMessageBox.warning(self, "Ошибка", f"Недостаточно количества блюда! Доступно только {available\_quantity}.")  
 return  
  
 # Оформляем заказ  
 total\_price = price \* quantity  
  
 # Начинаем транзакцию для обеспечения атомарности всех операций  
 conn = sqlite3.connect('restaurant.db')  
 cursor = conn.cursor()  
  
 try:  
 # Вставляем заказ в таблицу orders  
 cursor.execute(  
 "INSERT INTO orders (user\_login, dish\_id, quantity, total\_price, status) VALUES (?, ?, ?, ?, ?)",  
 (self.user\_login, dish\_id, quantity, total\_price, "Готовится"))  
  
 # Обновляем количество блюда в таблице dishes  
 new\_quantity = available\_quantity - quantity  
 cursor.execute("UPDATE dishes SET quantity = ? WHERE id = ?", (new\_quantity, dish\_id))  
  
 conn.commit() # Завершаем транзакцию  
 QMessageBox.information(self, "Успех", "Ваш заказ оформлен!")  
 self.load\_orders() # Обновляем список заказов  
 self.load\_dishes() # Обновляем список блюд  
  
 except Exception as e:  
 conn.rollback() # Если что-то пошло не так, откатываем изменения  
 QMessageBox.warning(self, "Ошибка", "Произошла ошибка при оформлении заказа: " + str(e))  
  
 finally:  
 conn.close()  
  
  
# --- Запуск приложения ---  
app = QApplication(sys.argv)  
login\_window = LoginWindow()  
login\_window.show()  
app.exec()