ДЕПАРТАМЕНТ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ТОМСКОЙ ОБЛАСТИ

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 «Информационные системы и программирование»

Разработка и тестирование информационной системы «Кофе пьём»

Пояснительная записка

к курсовому проекту

КП.23.09.02.07.602.08.ПЗ

Студент

«\_\_»\_\_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_ В. А. Кипоров

Руководитель

«\_\_»\_\_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_Д. А. Антипов

Томск 2023

Содержание

|  |
| --- |
| Введение 3 |
| 1 ОБЩАЯ ЧАСТЬ 5 |
| 1.1 Анализ предметной области 5 |
| 1.2 Выбор средств тестирования 8 |
| 2 СПЕЦИАЛЬНАЯ ЧАСТЬ 10 |
| 2.1 Описание требований к программе 10 |
| 2.2 Диаграмма вариантов использования 19 |
| 2.3 Диаграмма состояний 20 |
| 2.4 Схема данных 22 |
| 2.5 Тестовые сценарии 24 |
| ЗАКЛЮЧЕНИЕ 36 |
| Перечень используемых источников 37 |
| Приложение A. Результаты тестирования 38 |
| Приложение Б. Результаты работы программы 64 |
| Приложение Б. Листинг кода 68 |

Введение

Цель данного курсового проекта – разработать и протестировать приложение «Кофе пьём»

В приложении нужно реализовать: систему регистрации и авторизации, возможность просмотра профиля пользователя, системы смены пароля и удаления пользователей, способы просмотра меню и добавления блюд, функции создания и просмотра заказов, а также систему создания отчётов по продажам.

Во время тестирования будет использоваться методология чёрного ящика. Тестировщику не будет доступен исходный код проекта, благодаря чему можно провести наиболее приближенное к реальным условиям использования тестирование.

Задачи:

1. Провести анализ предметной области.

2. Разработать требования к информационной системе.

3. Описать тестовые случаи.

4. Подобрать наборы данных для тестирования.

5. Провести тестирование функций

1. ОБЩАЯ ЧАСТЬ
   1. Анализ предметной области

1.1.1 Характеристика предметной области

* Предметная область: общественное питание.
* Объект предметной области: продажа еды и напитков.
* Предмет автоматизации: процесс торговли питанием.

Проблема, которую решает информационная система: предприниматель ведёт учёт продаж, сотрудников и доходов в бумажном, малоэффективном формате.

Приложение для торговли едой должен автоматизировать следующие процессы:

* Учёт проданных блюд
* Учёт сотрудников
* Учёт финансов
* Создание заказов
* Создание отчётов

При запуске приложения пользователя встречает окно авторизации, в котором он должен ввести свои логин и пароль.

В системе существуют роли: администратор и продавец. Поэтому при авторизации в качестве администратора на главном окне отображены его фамилия и имя, существующие в системе заказы, зарегистрированные сотрудники и меню, а также реализованы возможности: создание отчётов, просмотр профиля, смена пароля, редактирование меню, добавление и удаление сотрудников и возможность деаутентификации. А при авторизации сотрудника в роли продавца, пользователь может наблюдать меню блюд и созданных заказов, так же в окне реализованы функции: создание заказов, просмотр профиля, в котором возможна смена пароля, и возможность деаутентификации.

1.1.2. Анализ конкурентов

1. Облачная информационная система «Presto»

Сайт: https://sbis.ru/presto

Плюсы:

* + Возможность ведения бухгалтерии
  + Кроссплатформенность
  + Использование карт сотрудников

Минусы:

* + Находится в стадии разработки

1. «Iiko»

Сайт: https://iiko.ru

Плюсы:

* + Скорость
  + Надёжность
  + Возможность дистанционного контроля за персоналом
  + Невозможность персонала совершать ошибки
  + Контроль поставщиков
  + Контроль аппаратуры заведения
  + Возможность управления несколькими заведениями

Минусы:

* + Разработчики игнорируют пожелания пользователей
  + Отсутствие мотивации сотрудников

1. Облачная информационная система «Poster»

Сайт: https://joinposter.com/

Плюсы:

* + Простой интерфейс
  + Отзывчивая поддержка
  + Удобный мониторинг финансов
  + Низкая цена в сравнении с конкурентами
  + Стабильность

Минусы:

* + Закрыт доступ к базе
  + Не работает на территории России и Белоруссии
  + Невозможность установки разных тарифов для разных заведений

1.1.3. Анализ целевой аудитории

Данный продукт будет интересен предпринимателям, открывшим заведение(я) общественного питания.

1.2 Выбор средств тестирования

Тестирование программного обеспечения — это метод проверки соответствия фактического программного продукта ожидаемым требованиям, который также необходим, чтобы убедиться, что продукт не содержит дефектов. Подразумевает выполнение предварительно определенных алгоритмов с использованием ручных или автоматизированных инструментов для оценки одного или нескольких интересующих свойств. Целью тестирования ПО является выявление ошибок, пробелов или отсутствующих требований, заданных на этапе проектирования продукта. В рамках данного курсового проекта необходимо провести ручное тестирование. Ручное тестирование программного обеспечения — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Ручное тестирование позволяет предварительно убедиться в том, что заявленный функционал работоспособен, не имеет ошибок, выдаёт ожидаемые, запланированные результаты и проверяется именно использование человеком, т.к. большинство проблем возникает именно у конечного потребителя.

Преимущества ручного тестирования:

* Простота и скорость тестирования.
* Возможность параллельной работы с заказчиком.

Недостатки ручного тестирования:

* Человеческий фактор. Разные тестировщики могут пройти один и тот же тест по-разному, но не протестировавши тот случай, который мог бы вызвать проблему.
* Неимение точного описания и понимания возникновения ошибки.

Тестирование будет проходить с использованием метода «чёрного ящика». Это метод тестирования программного продукта или приложения проходит без доступа к коду, базе данных и другим внутренним элементам. Для выполнения тестирования «чёрного ящика» нет необходимости доступа тестировщика к исходному коду. Тесты проводятся на основе возможного использования потребителем.

2. СПЕЦИАЛЬНАЯ ЧАСТЬ

2.1 Описание требований к программе

2.1.1 Общее описание

Программа «Кофе пьём» позволяет пользователю автоматизировать продажу продукции и управление бизнесом в сфере общественного питания.

Главная функция программы: создание заказов. У программы интуитивно понятный интерфейс.

Программа имеет функцию авторизации.

Есть возможность составления отчётов в Excel.

С помощью программы можно нанимать и увольнять сотрудников.

Программа позволяет добавлять блюда в меню.

После авторизации в роли администратора пользователь получает доступ к следующему функционалу:

* Просмотр своего профиля
* Смена пароля
* Просмотр заказов
* Создание отчётов
* Просмотр сотрудников
* Добавление и удаление сотрудников
* Просмотр меню
* Добавление и удаление блюд в меню
* Деаунтификация

После авторизации в роли продавца пользователь получает доступ к следующему функционалу:

* Просмотр своего профиля
* Смена пароля
* Просмотр блюд
* Создание заказов
* Просмотр заказов
* Деаунтификация

2.1.2 Требование к дизайну приложения

Приложение должно иметь интуитивно понятный интерфейс. Используется тёмная тема, что хорошо скажется на работе в ночное время суток. Фон не должен привлекать внимания, а текст «бить» по глазам. Кнопки, которые наиболее часто будут использоваться, должны быть более заметны нежели остальные.

2.1.3 Требования к представлению приложения

Первое окно (Авторизация) представляет собой страницу авторизации, где имеется два поля для ввода «Логин» и «Пароль», а также кнопка «Вход» по нажатию на которую происходит авторизация.

Следующее окно зависит от роли пользователя. Если пользователь зарегистрирован в роли администратора, то откроется «Окно администратора», а если пользователем является продавец, то откроется «Окно продавца».

«Окно администратора» содержит информацию о блюдах, работниках и меню, а также предоставляет возможности просмотра профиля, смены пароля, создания отчётов, добавления и удаления сотрудников, добавления и удаления блюд и деаунтификации.

Вверху окна содержатся такие компоненты как:

* Кнопка профиля
* Имя и фамилия
* Кнопка смены пароля
* Кнопка деаунтификации

Поле меню содержит в себе следующую информацию о блюдах:

* Номер блюда
* Наименование блюда
* Цена
* Кнопка удаления блюда

Поле сотрудники содержит следующую информацию пользователей:

* Номер сотрудника
* Имя
* Фамилия
* Отчество
* Отработанные часы
* Должность
* Кнопка удаления пользователя

Поле заказы отображает информацию о заказах:

* Номер заказа
* Цена
* Дата и время
* Название блюд в заказе, при нажатии на заказ

Внизу окна реализованы такие кнопки как:

* Создать отчёт
* Нанять сотрудника
* Добавить блюдо

«Окно продавца» содержит информацию о блюдах и заказах, а также предоставляет возможности создания заказов, просмотра профиля и деаунтификации.

Поле блюд содержит следующую информацию:

* Номер блюда
* Наименование
* Цена
* Желаемое количество
* Кнопка добавления блюда в предзаказ

В поле предзаказов содержится информация:

* Название блюда
* Стоимость за единицу
* Количество одинаковых блюд
* Кнопка удаления блюд из предзаказа

Поле заказы отображает информацию о заказах:

* Номер заказа
* Цена
* Дата и время создания
* Название блюд, при нажатии на заказ

Так же реализованы следующие кнопки:

* Заказать
* Кнопка профиля
* Кнопка деаунтификации

В каждом окне реализована кнопка профиля. При нажатии на эту кнопку открывается окно профиля «Профиль». В данном окне отображаются фамилия, имя и отчество авторизованного пользователя, а также в этом окне есть две кнопки: назад, которая возвращает пользователя к рабочему окну и сменить пароль, которая открывает окно смены пароля.

Так же в каждом окне реализована кнопка деаунтификации, которая возвращает пользователя в окно авторизации.

В окнах администратора и профиля реализована кнопка смены пароля. Она открывает окно смены пароля «Смена пароля», в котором отображены три поля ввода: текущий пароль, новый пароль и подтверждение нового пароля, а также в этом окне присутствуют две кнопки: «Сменить!», которая принимает изменения и возвращает пользователя в окно авторизации, и «Отмена», которая возвращает пользователя рабочему окну.

Кнопка «Создать отчёт», содержащаяся в окне администратора, создаёт отчёт по заказам, сортированных по месяцам, в excel таблицу.

Кнопка «Нанять сотрудника», реализованная в окне администратора, открывает окно «Добавление сотрудника», в котором присутствует кнопка «Нанять», которая создаёт пользователя с введёнными данными, а также следующие поля для ввода информации о новом пользователе:

* Логин
* Пароль
* Имя
* Фамилия
* Отчество
* Должность

Кнопка «Добавить блюдо», находящаяся в окне администратора, открывает окно «Добавление блюда», в котором присутствует кнопка добавить, при нажатии на которую создаётся блюдо, и два поля ввода информации о новом блюде:

* Название
* Цена

Кнопка удаления пользователя, реализована в виде иконки крестика в списке «Сотрудники». При нажатии на эту кнопку удалятся пользователь, описанный напросив этой кнопки.

В окне продавца реализована кнопка добавления блюда в предзаказ. Она находится в списке блюд и добавляет блюдо, находящееся напротив, в предзаказ.

Кнопка «Заказать», находящаяся в окне продавца создаёт заказ из блюд, находящихся в предзаказе.

2.1.4 Требования к структуре приложения

Проект содержит следующие разделы:

* Окно авторизации
* Окно продавца
* Окно администратора
* Окно профиля
* Окно смены пароля
* Окно добавления пользователя
* Окно добавления блюда

2.1.5 Требования к хранению данных

Приложение предполагает ввод, хранение и обработку данных. Для хранения данных используется база данных – PostgreSQL 15.

2.1.6 Требования к языкам программирования

Приложение реализовано с использованием языка C# и фреймворка Avalonia. При использовании C# возможно использование объектно-ориентированного подхода программирования, а Avalonia позволяет создать красивый дизайн приложения и при необходимости сделать его кроссплатформенным.

2.1.7 Требования к иллюстрациям

Все иллюстрации соответствуют форматам png.

2.1.8 Требования к программному обеспечению серверной части

Для корректного функционирования приложения необходимо следующее программное обеспечение:

* Операционная система – Windows 8 и выше.

2.1.9 Требования к техническому обеспечению

Для работы приложения необходимо техническое обеспечение со следующими минимальными характеристиками:

* Процессор - c частотой 1 гигагерц (ГГц) или выше.
* Оперативная память - 2 ГБ (для 32-разрядной системы) или 3 ГБ (для 64-разрядной системы).
* Место на жестком диске - 25 ГБ (для 32-разрядной системы) или 30 ГБ (для 64-разрядной системы).

2.1.10 Требования к лингвистическому обеспечению

Интерфейс приложения использует русский язык.

2.1.11 Требования к адаптивности приложения

Приложение корректно отображается на мониторах с разрешением 1080х720 и больше

2.2 Диаграмма вариантов использования

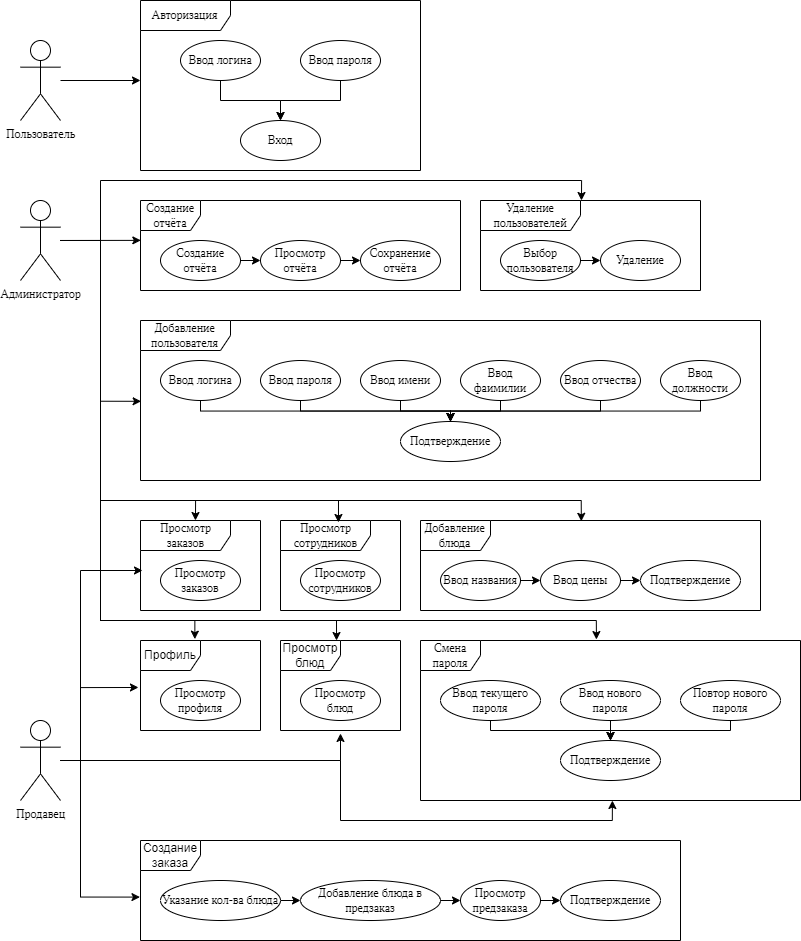


Рисунок 1 - Диаграмма вариантов использования

2.3 Диаграмма состояний

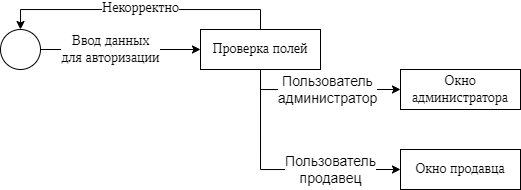


Рисунок 2 - Диаграмма состояний авторизации

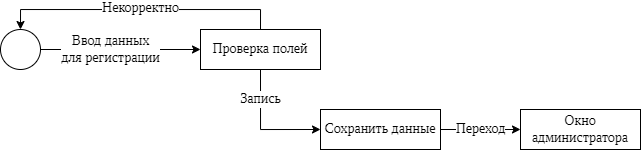


Рисунок 3 - Диаграмма состояний добавления нового пользователя

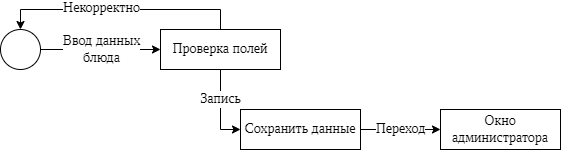


Рисунок 4 - Диаграмма состояний добавление нового блюда



Рисунок 5 - Диаграмма состояний удаление пользователя

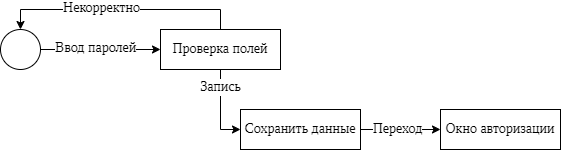


Рисунок 6 - Диаграмма состояний смены пароля

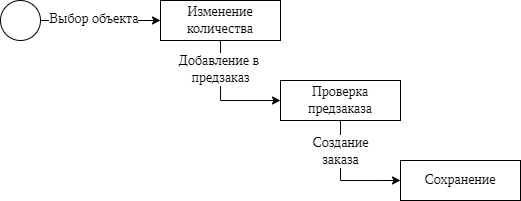


Рисунок 7 - Диаграмма состояний создания заказа

2.4 Схема данных

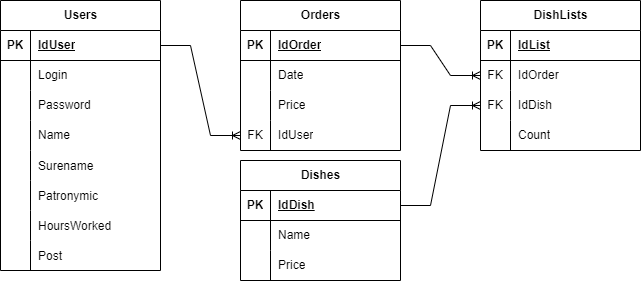


Рисунок 8. Схема базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Названия поля | Тип данных | Описание |
| 1 | IdUser | INT | Autoincrement, PK  Уникальный  идентификатор пользователя |
| 2 | Login | STRING | Логин пользователя |
| 3 | Password | STRING | Пароль пользователя |
| 4 | Name | STRING | Имя пользователя |
| 5 | Surename | STRING | Отчество пользователя |
| 6 | Patronymic | STRING | Фамилия пользователя |
| 7 | HoursWorked | DOUBLE | Отработанные часы пользователя |
| 8 | Post | STRING | Должность пользователя |

Таблица 1 – Словарь данных таблицы Users

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Названия поля | Тип данных | Описание |
| 1 | IdOrder | INT | Autoincrement, PK  Уникальный идентификатор заказа |
| 2 | IdUser | INT | FK, ссылка на Id пользователя |
| 3 | Date | DATETIME | Дата создания заказа |
| 4 | Price | FLOAT | Цена заказа |

Таблица 2 – Словарь данных таблицы Orders

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Названия поля | Тип данных | Описание |
| 1 | IdDish | INT | Autoincrement, PK, Уникальный идентификатор блюда |
| 2 | Name | STRING | Наименование блюда |
| 3 | Price | FLOAT | Цена блюда |

Таблица 3 – Словарь данных таблицы Dishes

|  |  |  |  |
| --- | --- | --- | --- |
| № п-п | Названия поля | Тип данных | Описание |
| 1 | IdList | INT | Autoincrement, PK |
| 2 | IdOrder | INT | FK, ссылка на Id заказа |
| 3 | IdDish | INT | FK, ссылка на Id блюда |
| 4 | Count | INT | Количество одинаковых блюд |

Таблица 4 – Словарь данных таблицы DishList

2.5 Тестовые сценарии

Тестовый пример #1:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_1 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Тест регистрации пользователя. |
| Краткое изложение теста | Проверка вводимых данных для регистрации Условия:  1. Логин и пароль должны быть заполнены  2. Логины разных пользователей не должны совпадать |
| Этапы теста | 1. Ввод необходимых данных  2. Проверка данных на валидность |
| Тестовые данные | 1. Логин: log, пароль: pass, имя: Иван, фамилия: Иванов, отчество: Иванович, должность: Администратор.  2. Логин: «», пароль: pass, имя: Иван, фамилия: Иванов, отчество: Иванович, должность: Администратор.  3. Логин: log, пароль: «», имя: Иван, фамилия: Иванов, отчество: Иванович, должность: Администратор.  4. Логин: log, пароль: pass, имя: «», фамилия: Иванов, отчество: Иванович, должность: Администратор.  5. Логин: log, пароль: pass, имя: Иван, фамилия: «», отчество: Иванович, должность: Администратор.  6. Логин: 1, пароль: 1, имя: Иван, фамилия: Иванов, отчество: «», должность: Администратор.  7. Логин: 1, пароль: 1, имя: Иван, фамилия: Иванов, отчество: Иванович, должность: «».  8. Логин: «», пароль: «», имя: Иван, фамилия: Иванов, отчество: Иванович, должность: Администратор.  9. Логин: «», пароль: «», имя: «», фамилия: Иванов, отчество: Иванович, должность: Администратор.  10. Логин: «», пароль: «», имя: «», фамилия: «», отчество: Иванович, должность: Администратор.  11. Логин: «», пароль: «», имя: «», фамилия: «», отчество: «», должность: Администратор.  12. Логин: «», пароль: «», имя: «», фамилия: «», отчество: «», должность: «».  13. Логин: 1, пароль: 1, имя: Иван, фамилия: Иванов, отчество: Иванович, должность: Администратор. |
| Ожидаемый результат | 1. Пройден.  2. Не пройден.  3. Не пройден.  4. Не пройден.  5. Не пройден.  6. Пройден.  7. Не пройден.  8. Не пройден.  9. Не пройден.  10. Не пройден.  11. Не пройден.  12. Не пройден.  13. Не пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыта страница добавления сотрудников «Добавление сотрудника». |
| Постусловие | Регистрация нового пользователя. |
| Примечание/комментарии | Протестированы следующие случаи:  1. Все поля заполнены верно.  2. Заполнены все поля кроме логина.  3. Заполнены все поля кроме пароля.  4. Заполнены все поля кроме имени.  5. Заполнены все поля кроме фамилии.  6. Заполнены все поля кроме отчества.  7. Заполнены все поля кроме должности.  8. Заполнены все поля кроме логина и пароля.  9. Заполнены все поля кроме логина, пароля и имени.  10. Заполнены все поля кроме логина, пароля, имени и фамилии.  11. Заполнена только должность.  12. Все поля пусты.  13. Все поля заполнены, но пользователь с таким логином уже зарегистрирован. |

Тестовый пример #2:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_2 |
| Приоритет тестирования | Высокий. |
| Заголовок/название теста | Тестирование авторизации пользователя. |
| Краткое изложение теста | Проверка вводимых данных для авторизации. Условия:  1. Логин и пароль должны быть заполнены.  2. Пользователь должен быть зарегистрирован. |
| Этапы теста | 1. Ввод необходимых данных.  2. Проверка данных на валидность. |
| Тестовые данные | 1. Логин: 1, Пароль: 1;  2. Логин: log, Пароль: password;  3. Логин: «», Пароль: 1.  4. Логин: 1, Пароль: «»;  5. Логин: «», Пароль: «». |
| Ожидаемый результат | 1. Пройден.  2. Не пройден.  3. Не пройден.  4. Не пройден.  5. Не пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыта страница авторизации, пользователь был зарегистрирован. |
| Постусловие | Открытие страницы магазина. |
| Примечание/комментарии | Протестированы следующие случаи:  1. Ввод логина и пароля зарегистрированного пользователя.  2. Ввод несуществующего логина.  3. Пустое поле для логина.  4. Пустое поля для пароля.  5. Пустые оба поля. |

Тестовый пример #3:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_3 |
| Приоритет тестирования | Высокий. |
| Заголовок/название теста | Добавление блюда. |
| Краткое изложение теста | Проверка вводимых данных для добавления. Условия:  1. Имя и цена должны быть заполнены. |
| Этапы теста | 1. Открытие окна добавление блюда «Добавление блюда».  2. Ввод необходимых данных. |
| Тестовые данные | 1. Название: булочка, цена: 30,5.  2. Название: «», цена: 30,5.  3. Название: булочка, цена: «».  4. Название: «», цена: «». |
| Ожидаемый результат | 1. Пройден.  2. Не пройден.  3. Не пройден.  4. Не пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыто окно добавления блюда «Добавить блюдо». |
| Постусловие | Добавлено новое блюдо. |
| Примечание/комментарии | Протестированы следующие случаи:  1. Заполнение всех полей.  2. Заполнение всех полей, кроме названия.  3. Заполнение всех полей, кроме цены.  4. Не заполнено ни одно поле. |

Тестовый пример #4:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_4 |
| Приоритет тестирования | Средний. |
| Заголовок/название теста | Изменение пароля. |
| Краткое изложение теста | Проверка вводимых данных для смены пароля. Условия:  1. Новый пароль не должен совпадать текущим.  2. Все поля должны быть заполнены.  3. Поля «новый пароль» и «повторите пароль» должны совпадать.  4. Поле «текущий пароль» должно совпадать с паролем авторизованного пользователя. |
| Этапы теста | 1. Открытие окна смены пароля «Смена пароля».  2. Ввод необходимых данных. |
| Тестовые данные | 1. Текущий пароль: 1, новый пароль: 12, повторите пароль: 12.  2. Текущий пароль: 1, новый пароль: 1, повторите пароль: 1.  3. Текущий пароль: «», новый пароль: 1, повторите пароль: 1.  4. Текущий пароль: 12, новый пароль: 1, повторите пароль: 1.  5. Текущий пароль: 1, новый пароль: «», повторите пароль: 12.  6. Текущий пароль: 1, новый пароль: 12, повторите пароль: «».  7. Текущий пароль: «», новый пароль: «», повторите пароль: «». |
| Ожидаемый результат | 1. Пройден.  2. Пройден.  3. Не пройден.  4. Не пройден.  5. Не пройден.  6. Не пройден.  7. Не пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Не зачёт. |
| Предварительное условие | Открыто окно смены пароля «Смена пароля». |
| Постусловие | Изменение пароля пользователя. |
| Примечание/комментарии | Протестированы следующие случаи:  1. Заполнение всех полей, текущий пароль и поле «текущий пароль» совпадают, текущий пароль и поле «новый пароль» не совпадают, поле «новый пароль» и поле «повторите пароль» совпадают.  2. Заполнение всех полей, текущий пароль и поле «текущий пароль» совпадают, текущий пароль и поле «новый пароль» совпадают, поле «новый пароль» и поле «повторите пароль» совпадают.  3. Заполнение полей только «новый пароль» и «повторите пароль», поле «новый пароль» и поле «повторите пароль» совпадают.  4. Заполнение всех полей, текущий пароль и поле «текущий пароль» не совпадают, текущий пароль и поле «новый пароль» не совпадают, поле «новый пароль» и поле «повторите пароль» совпадают.  5. Заполнение полей только «текущий пароль» «повторите пароль», текущий пароль и поле «текущий пароль» совпадают.  6. Заполнение полей только «текущий пароль» и «новый пароль», текущий пароль и поле «текущий пароль» совпадают, текущий пароль и поле «новый пароль» не совпадают.  7. Все поля пустые. |

Тестовый пример #5:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_5 |
| Приоритет тестирования | Высокий. |
| Заголовок/название теста | Создание заказа. |
| Краткое изложение теста | Проверка вводимых данных для создания заказа  Условия:  1. Должно быть открыто окно продавца.  2. Количество одинаковых блюд не равно нулю. |
| Этапы теста | 1. Изменить количество блюд, если требуется.  2. Нажать кнопку добавления в предзаказ.  3. Нажать кнопку «Заказать». |
| Тестовые данные | 1. Номер блюда: 1, название: пончик, цена: 50, количество: 1.  2. Номер блюда: 4, название: раф, цена: 70, количество: 0. |
| Ожидаемый результат | 1. Пройден.  2. Не пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыта главная страница. |
| Постусловие | Сортировка товара. |
| Примечание/комментарии | Протестированы следующие случаи:  1. Выбранное блюдо с параметром количества «1» добавлено.  2. Выбранное блюдо с параметром количества «0» не добавлено в предзаказ. |

Тестовый пример #6:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_6 |
| Приоритет тестирования | Средний. |
| Заголовок/название теста | Создание отчётов. |
| Краткое изложение теста | Для создания отчёта авторизуемся как администратор и нажимаем кнопку создать отчёт, после чего должен создаться excel файл. |
| Этапы теста | 1. Авторизуемся как администратор.  2. Нажать кнопку «Создать отчёт». |
| Тестовые данные | Данные для авторизации:   * Логин: 1. * Пароль: 1.   Нажатие на кнопку «Создать отчёт». |
| Ожидаемый результат | Пройден. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыта главная страница администратора. |
| Постусловие | Создание excel таблицы. |
| Примечание/комментарии | После создания файла необходимо сохранить его. |

Тестовый пример #7:

|  |  |
| --- | --- |
| Тестовый пример# | ТП\_П\_7 |
| Приоритет тестирования | Средний. |
| Заголовок/название теста | Тестирование удаления пользователя. |
| Краткое изложение теста | Проверка удаления пользователей. Для этого необходимо в окне администратора в поле «Сотрудники» нажать на «крестик» напротив удаляемого сотрудника. |
| Этапы теста | 1. Авторизация в роли администратора  2. Нажатие на «крестик» напротив удаляемого сотрудника. |
| Тестовые данные | Удаляем сотрудника с данными   * №: 24. * Имя: Test. * Фамилия: Delete. * Отчество: employee . |
| Ожидаемый результат | Данный пользователь будет удалён. |
| Фактический результат | Совпадает с ожидаемым. |
| Статус | Зачёт. |
| Предварительное условие | Открыта страница администратора. |
| Постусловие | Пользователь удалён. |
| Примечание/комментарии | Нет. |

Заключение

В данном курсовом проекте было разработано и протестировано приложение «Кофе пьём». Проанализирована предметная область, составлены тест-кейсы для тестируемых функций. Реализованы следующие функции:

* Авторизация
* Добавление пользователей
* Удаление пользователей
* Вывод блюд
* Вывод заказов
* Создание заказов
* Просмотр профиля
* Смена пароля
* Создание отчётов
* Добавление блюд

Таким образом, главная цель и задачи курсового проекта по разработке и тестированию приложения «Кофе пьём» были выполнены.

Перечень используемых источников

1. Avalonia UI. Документация: [Электронный ресурс]. – Режим доступа: https://avaloniaui.net
2. Microsoft. Документация: [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp
3. Metanit. Форум: [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp>
4. Stack Overflow. Форум: [Электронный ресурс]. – Режим доступа: https://stackoverflow.com/questions/tagged/c%23

Результаты тестирования

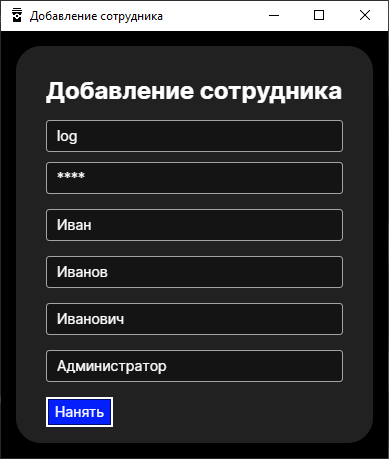


Рисунок А. 1.1 - Тест регистрации пользователя. Ввод данных

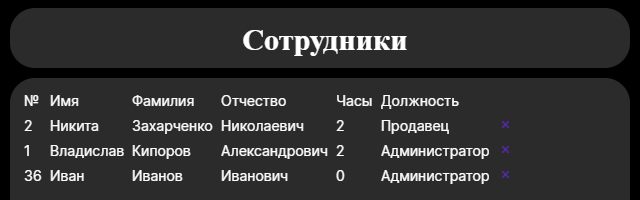


Рисунок А. 1.2 - Пользователь зарегистрирован. Результат

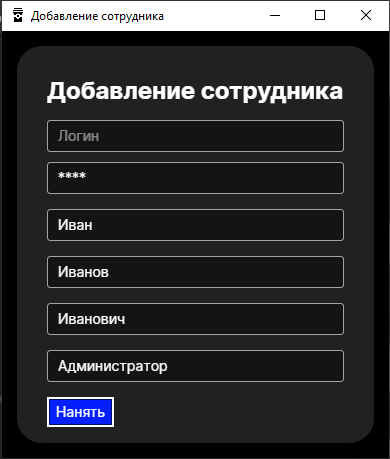


Рисунок А. 1.3 - Тест регистрации пользователя. Ввод данных

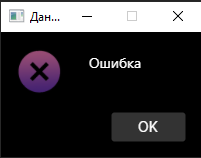


Рисунок А. 1.4 - Тест регистрации пользователя. Результат

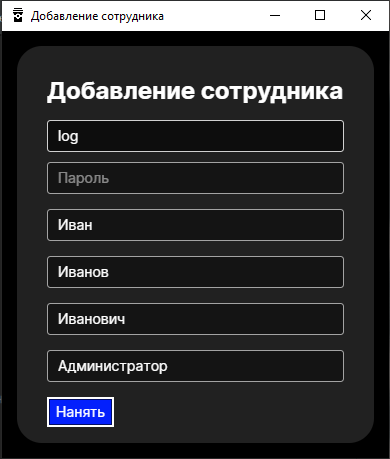


Рисунок А. 1.5 - Тест регистрации пользователя. Ввод данных

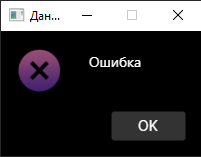


Рисунок А 1.6 - Тест регистрации пользователя. Результат

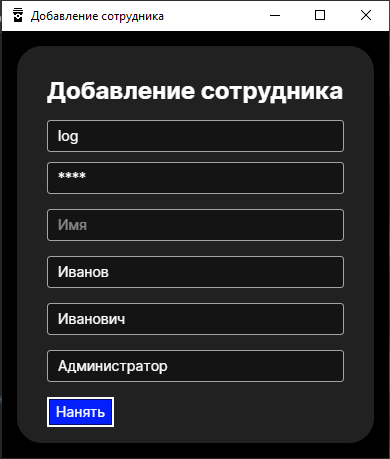


Рисунок А. 1.7 - Тест регистрации пользователя

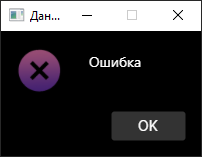


Рисунок А. 1.8 - Тест регистрации пользователя

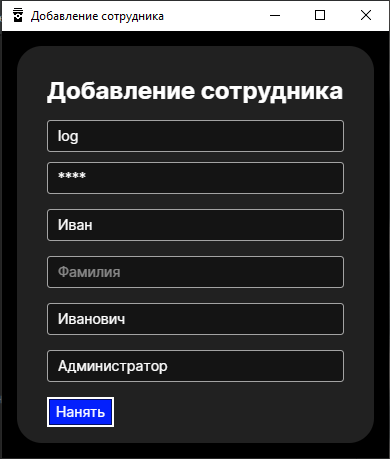


Рисунок А. 1.9 - Тест регистрации пользователя

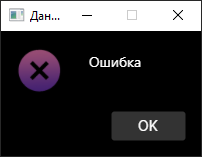


Рисунок А. 1.10 - Тест регистрации пользователя

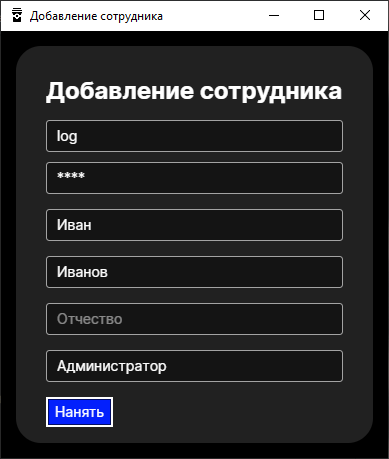


Рисунок А. 1.11 - Тест регистрации пользователя



Рисунок А. 1.12 - Тест регистрации пользователя

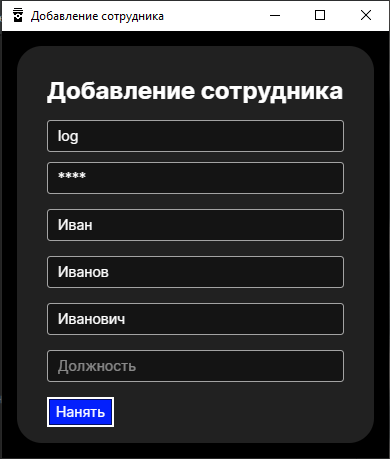


Рисунок А. 1.13 - Тест регистрации пользователя

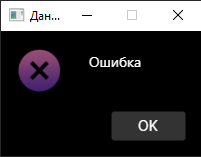


Рисунок А. 1.14 - Тест регистрации пользователя

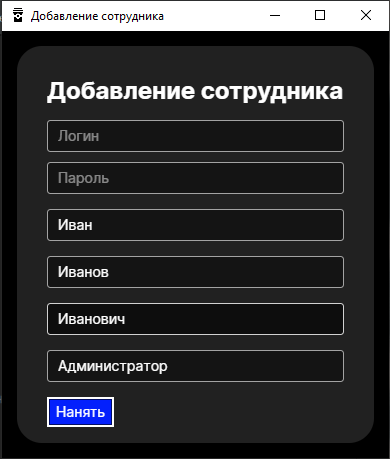


Рисунок А. 1.15 - Тест регистрации пользователя

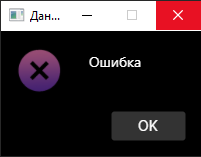


Рисунок А. 1.16 - Тест регистрации пользователя

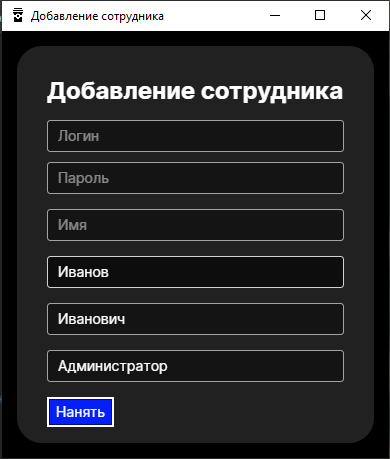


Рисунок А. 1.17 - Тест регистрации пользователя

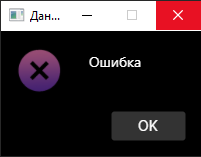


Рисунок А. 1.18 - Тест регистрации пользователя

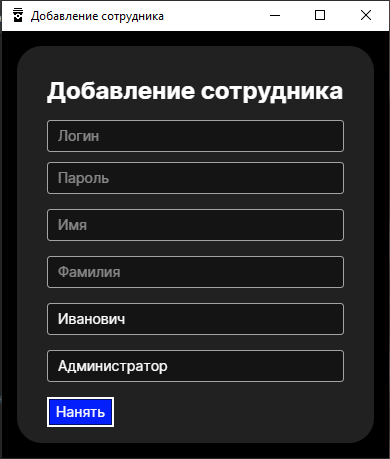


Рисунок А. 1.19 - Тест регистрации пользователя

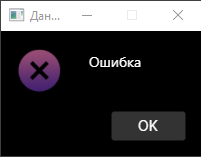


Рисунок А. 1.20 - Тест регистрации пользователя

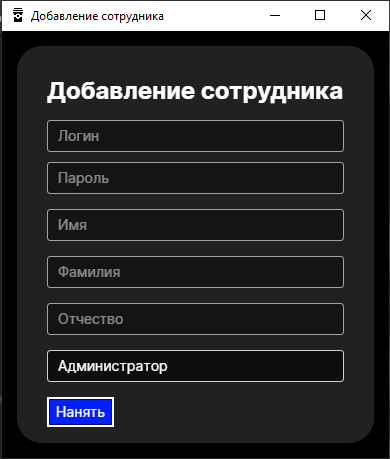


Рисунок А. 1.21 - Тест регистрации пользователя

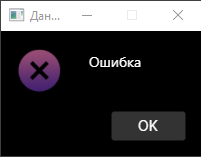


Рисунок А. 1.22 - Тест регистрации пользователя

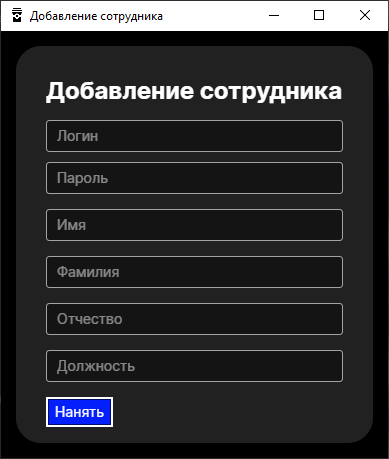


Рисунок А. 1.23 - Тест регистрации пользователя

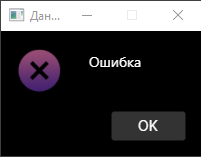


Рисунок А. 1.24 - Тест регистрации пользователя

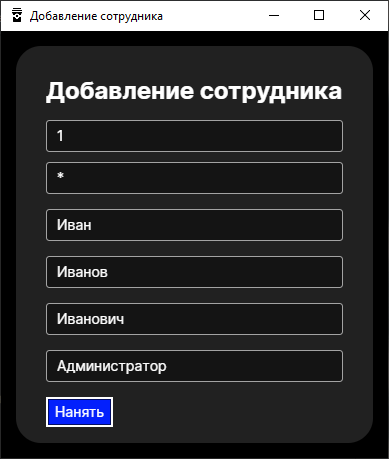


Рисунок А. 1.25 - Тест регистрации пользователя

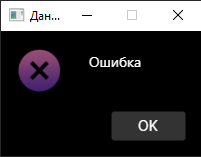


Рисунок А. 1.26 - Тест регистрации пользователя

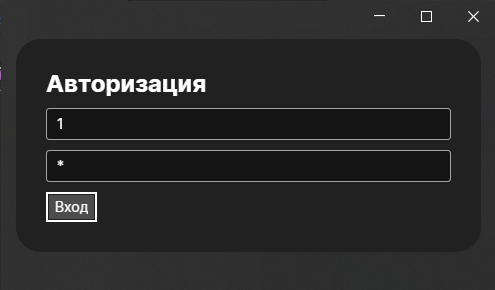


Рисунок А. 2.1 - Тест авторизации пользователя. Ввод данных

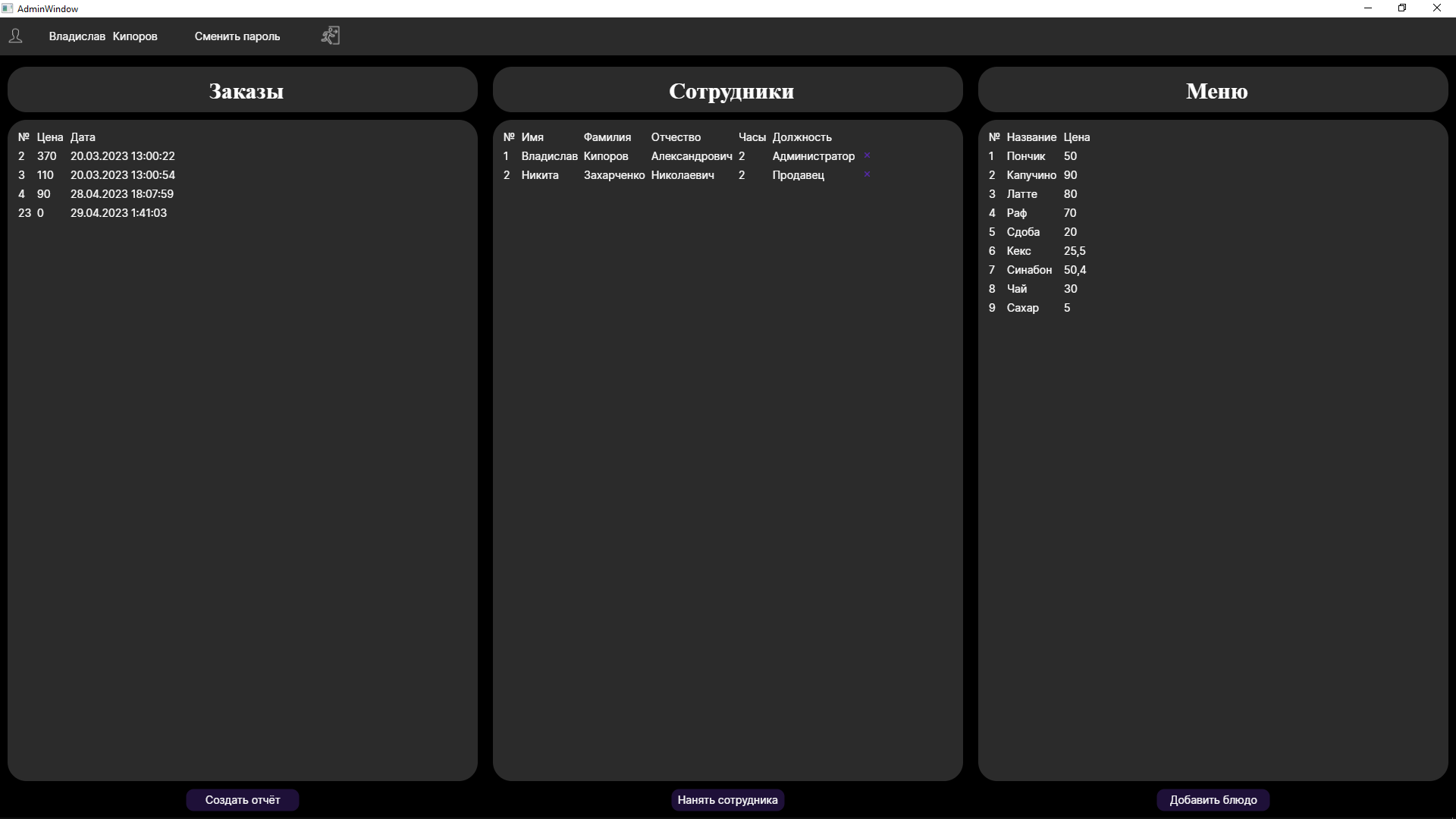


Рисунок А. 2.2 - Тест авторизации пользователя. Результат

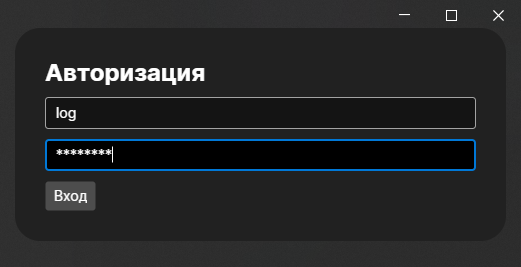


Рисунок А. 1.3 - Тест авторизации пользователя. Ввод данных

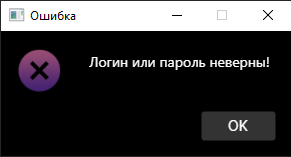


Рисунок А. 2.4 - Тест авторизации пользователя. Результат

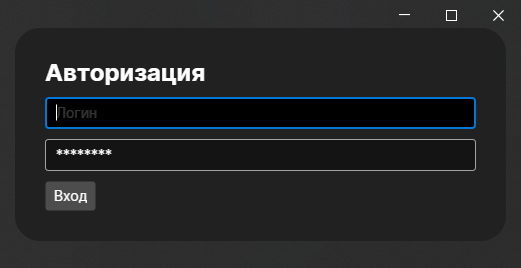


Рисунок А. 2.5 - Тест авторизации пользователя. Ввод данных

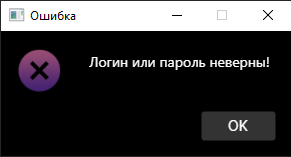


Рисунок А. 2.6 - Тест авторизации. Результат

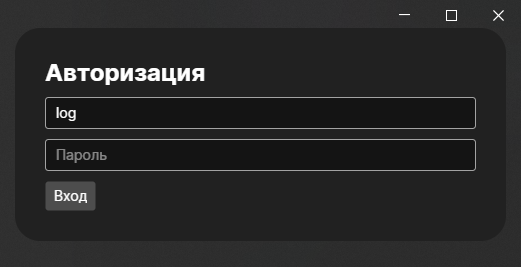


Рисунок А. 2.7 - Тест авторизации пользователя. Ввод данных

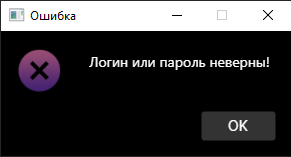


Рисунок А. 2.8 - Тест авторизации пользователя. Результат

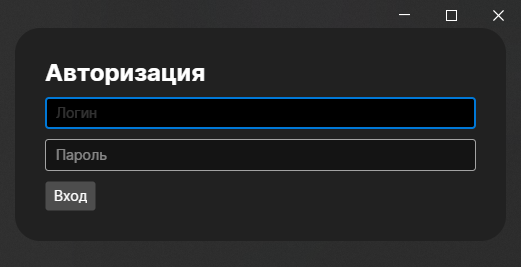


Рисунок А. 2.9 - Тест авторизации пользователя. Ввод данных

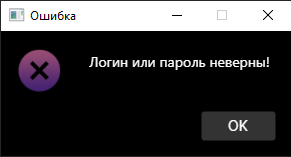


Рисунок А. 2.10 - Тест авторизации пользователя. Результат

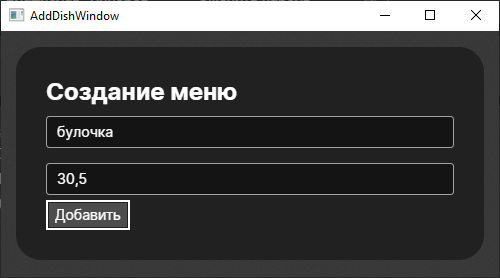


Рисунок А. 3.1 - Тест добавления блюда. Ввод данных



Рисунок А. 3.2 - Тест добавления блюда. Результат

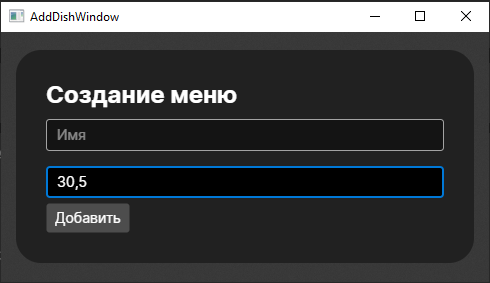


Рисунок А. 3.3 - Тест добавления блюда. Ввод данных

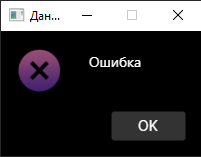


Рисунок А. 3.4 - Тест добавления блюда. Результат

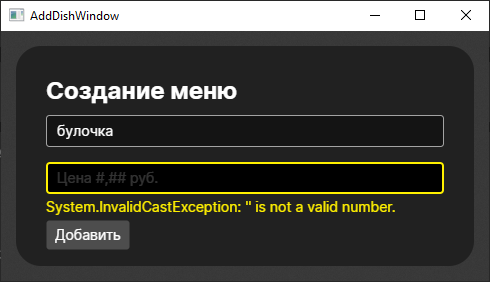


Рисунок А. 3.5 - Тест добавления блюда. Ввод данных

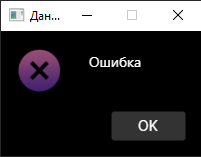


Рисунок А. 3.6 - Тест добавления блюда. Результат

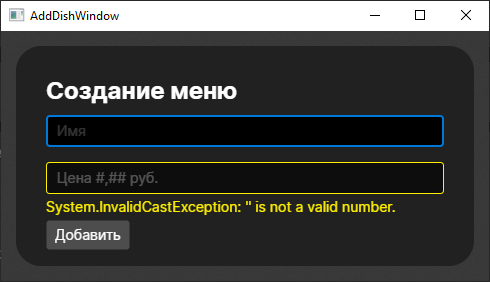


Рисунок А. 3.7 - Тест добавления блюда. Ввод данных

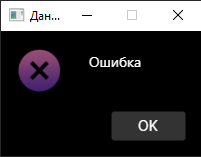


Рисунок А. 3.8 - Тест добавления блюда. Результат

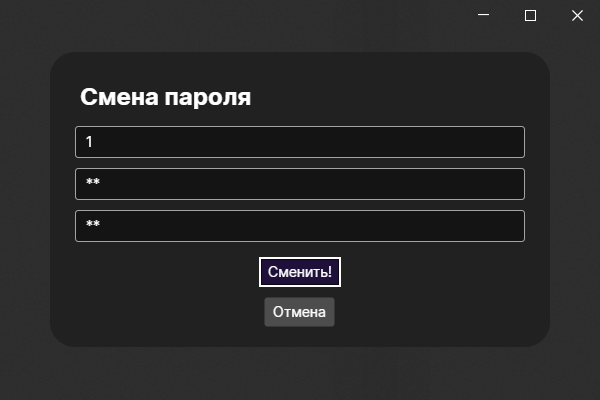


Рисунок А. 4.1 - Тест смены пароля. Ввод данных

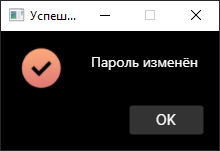


Рисунок А. 4.2 - Тест смены пароля. Результат

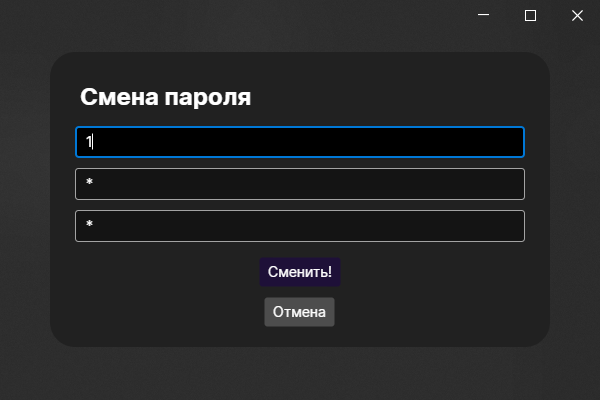


Рисунок А. 4.3 - Тест смены пароля. Ввод данных

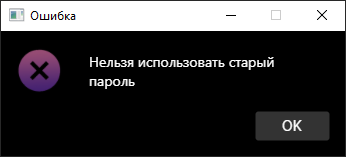


Рисунок А. 4.4 - Тест смены пароля. Результат

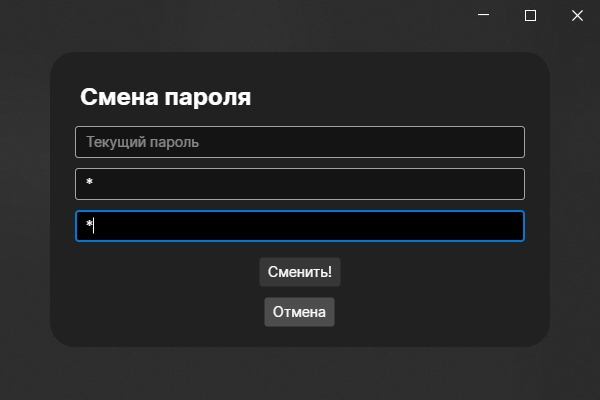


Рисунок А. 4.5 - Тест смены пароля. Ввод данных

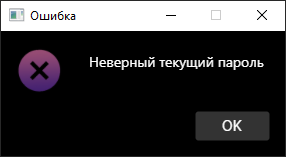


Рисунок А. 4.6 - Тест смены пароля. Результат

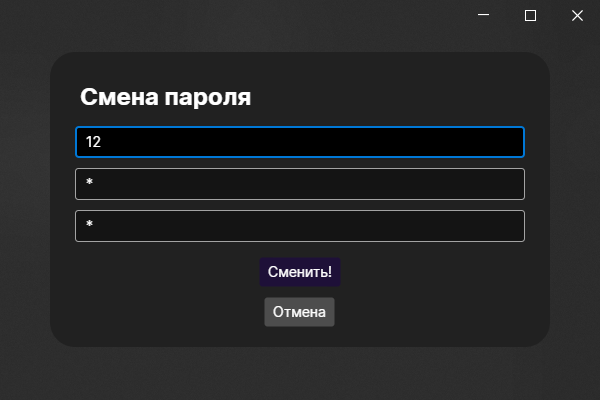


Рисунок А. 4.7 - Тест смены пароля. Ввод данных

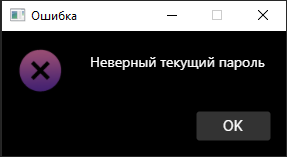


Рисунок А. 4.8 - Тест смены пароля. Результат

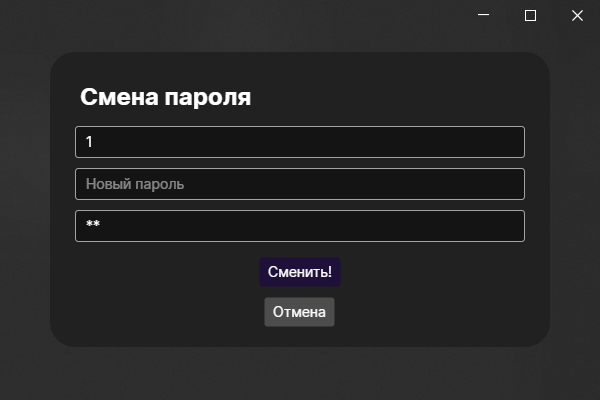


Рисунок А. 4.9 - Тест смены пароля. Ввод данных

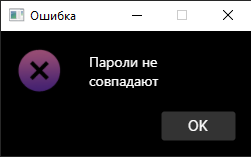


Рисунок А. 4.10 - Тест смены пароля. Результат

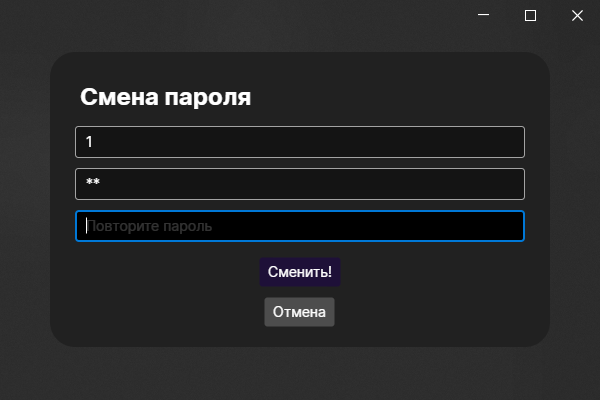


Рисунок А. 4.11 - Тест смены пароля. Ввод данных

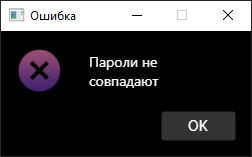


Рисунок А. 4.12 - Тест смены пароля. Результат

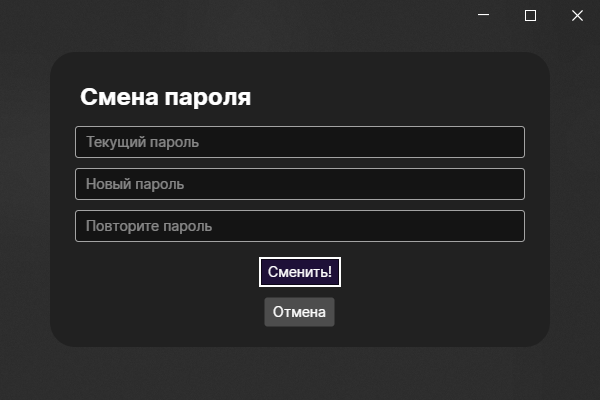


Рисунок А. 4.13 - Тест смены пароля. Ввод данных

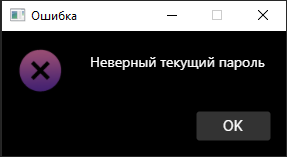


Рисунок А. 4.14 - Тест смены пароля. Результат

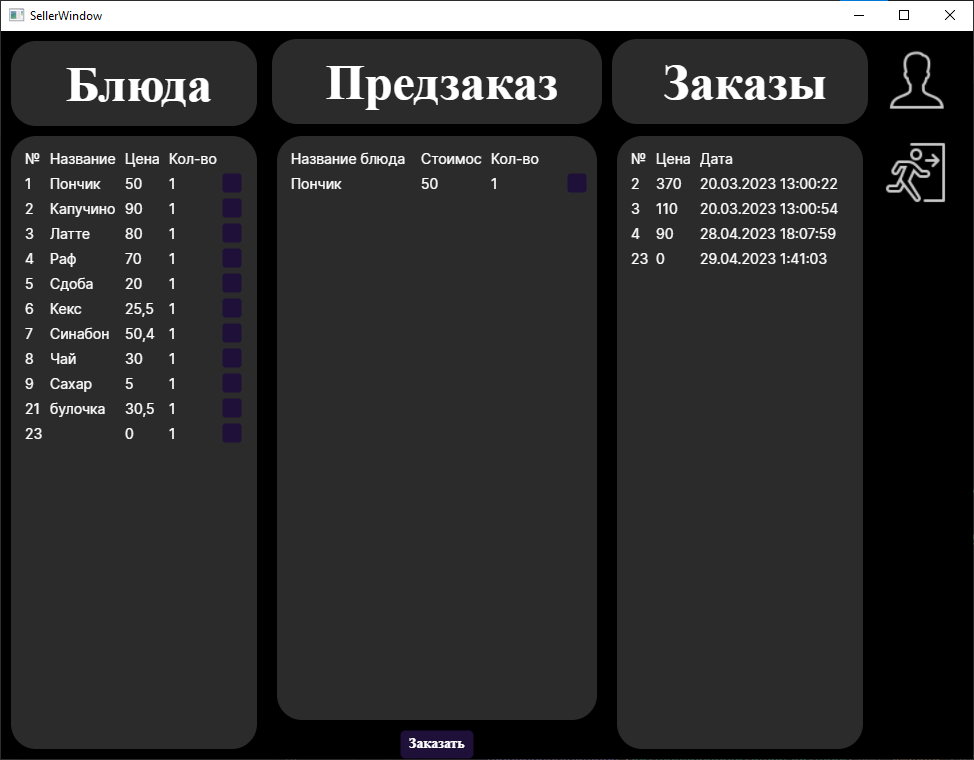


Рисунок А. 5.1 - Создание заказа. Добавление блюда в предзаказ

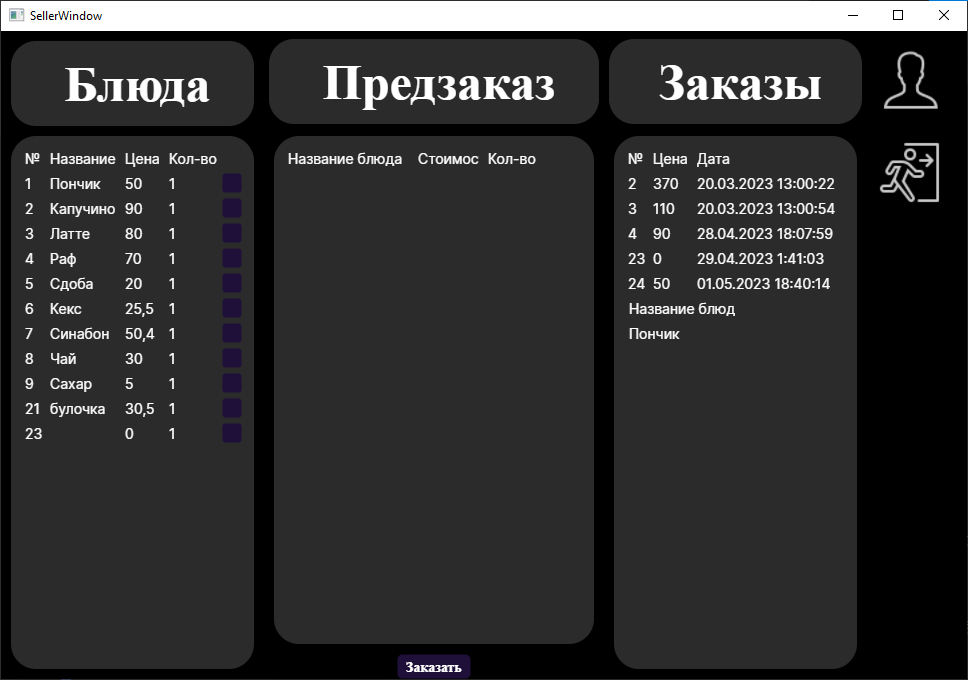


Рисунок А. 5.2 - Создание заказа. Результат

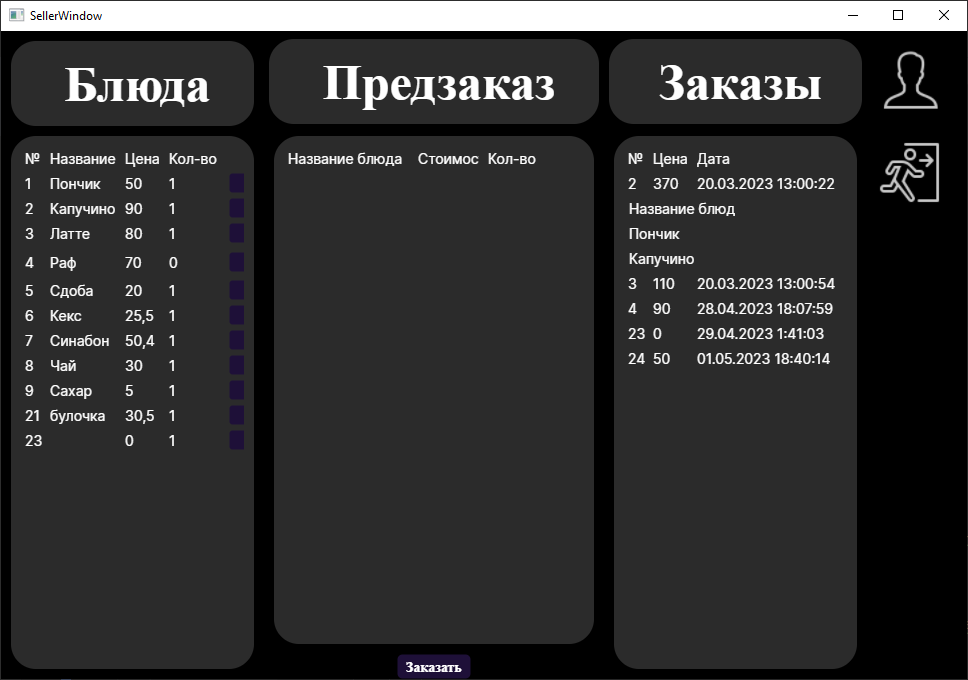


Рисунок А. 5.3 - Добавление блюда в предзаказ невозможно

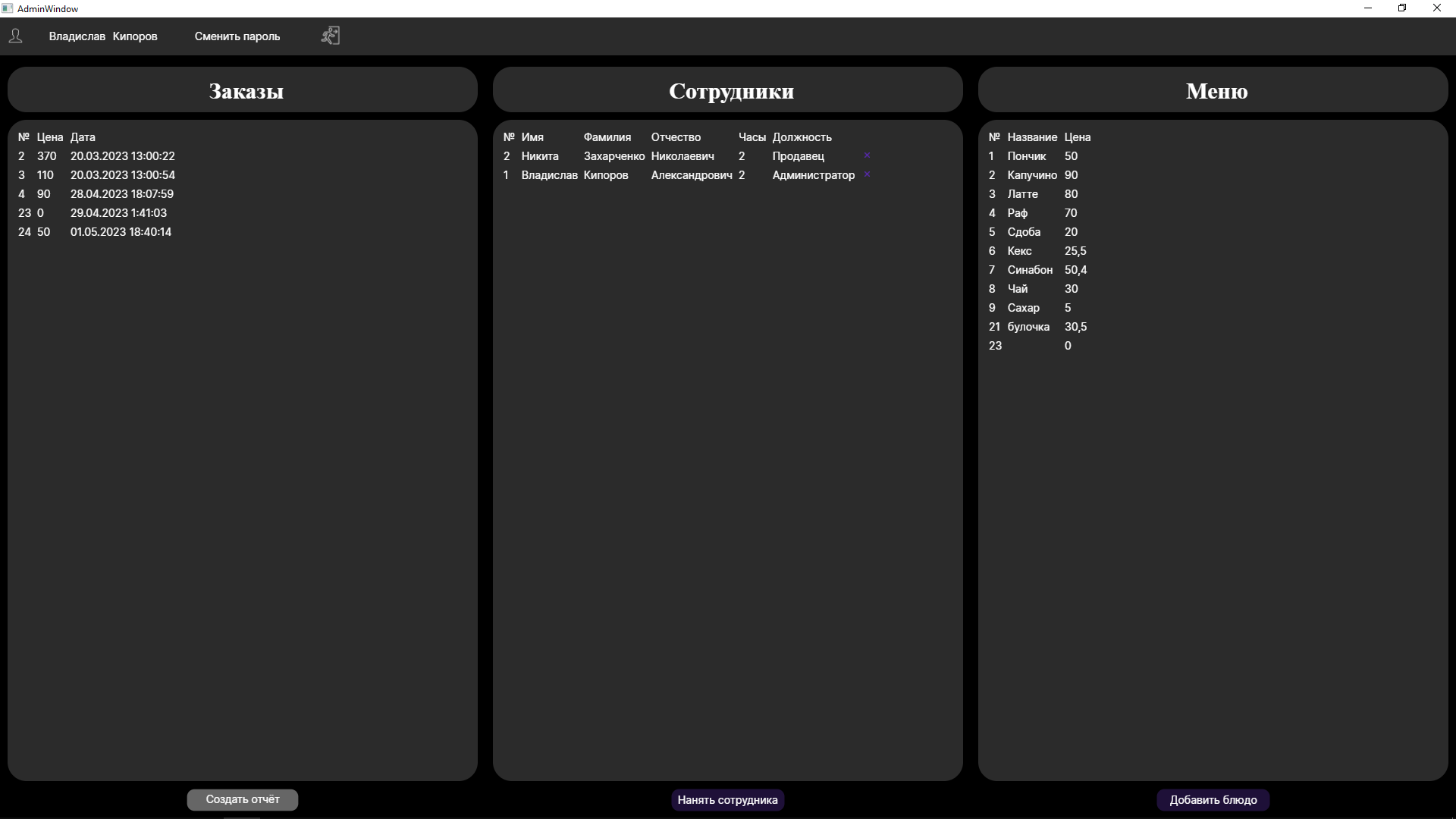


Рисунок А. 5.4 – Создание отчёта. Нажатие кнопки «Создать отчёт»

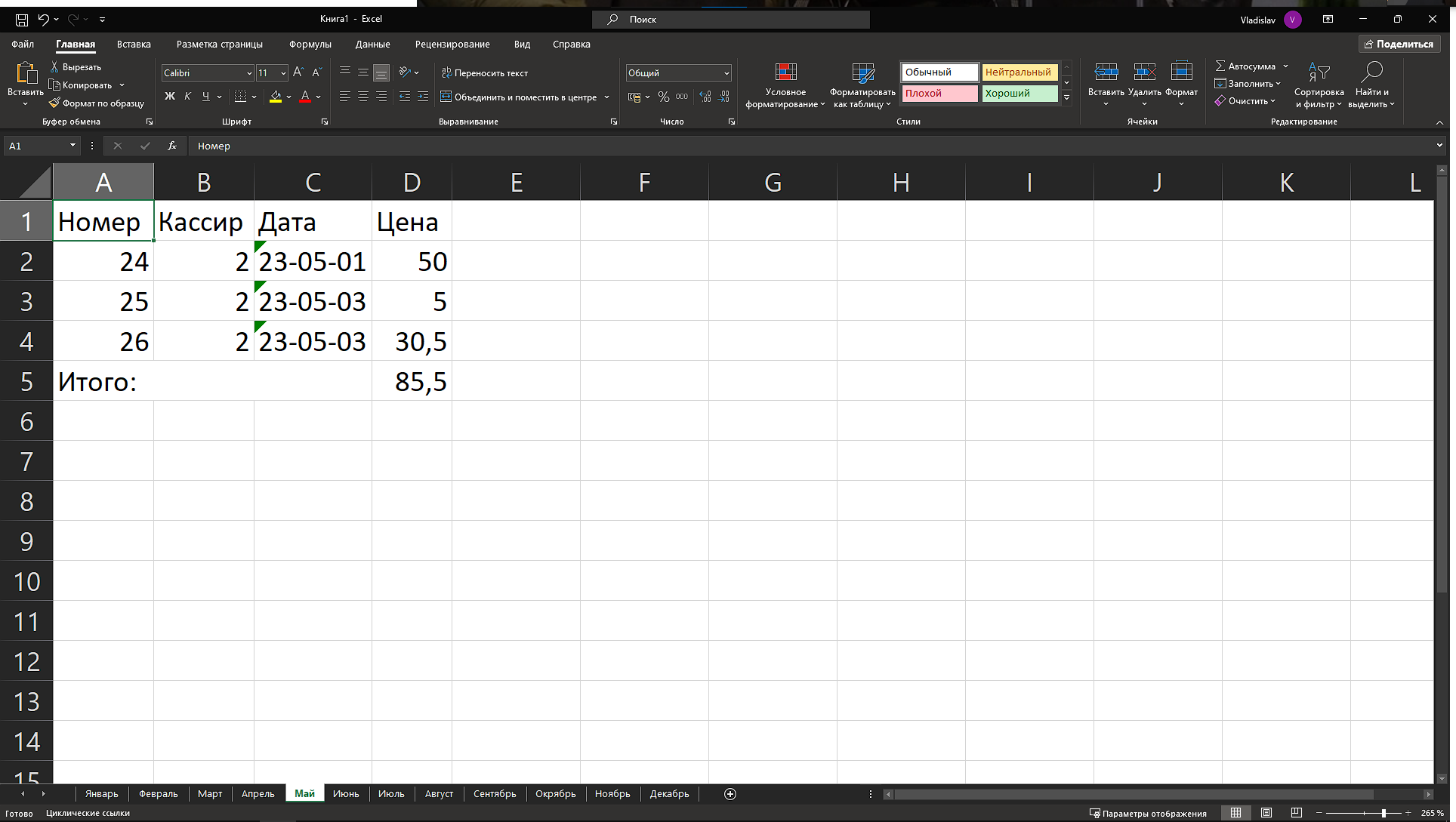


Рисунок А. 5.5 – Создание отчёта. Результат

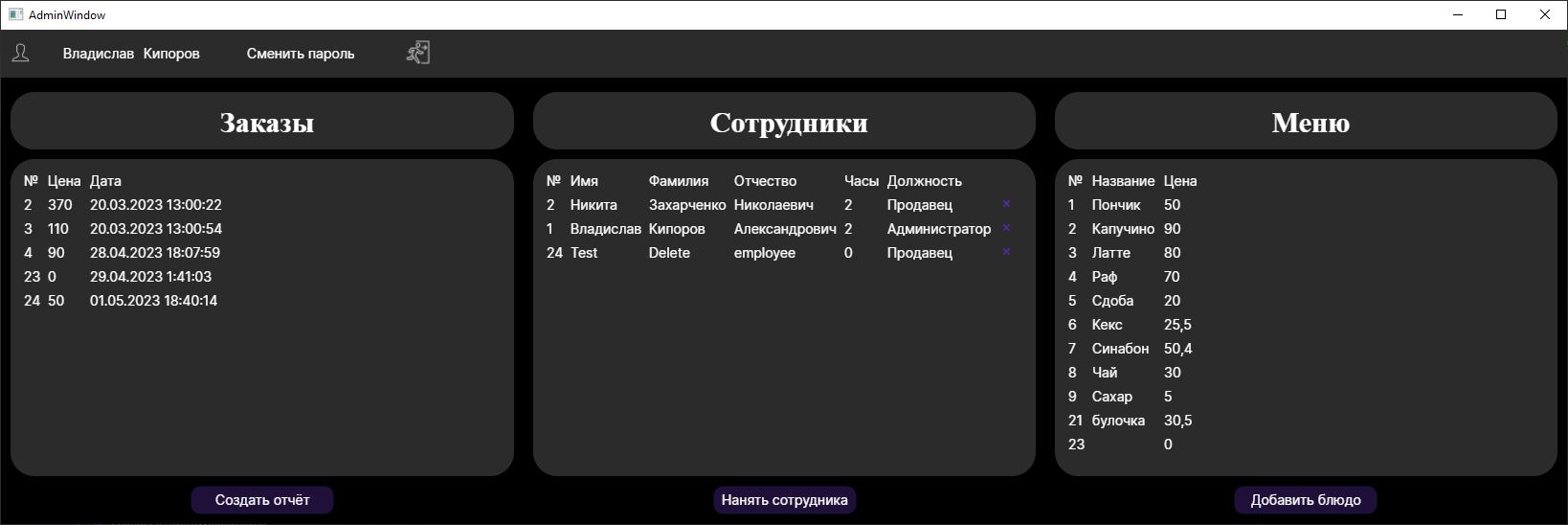


Рисунок А. 6.1 - Тест удаления пользователей. Состояние до удаления

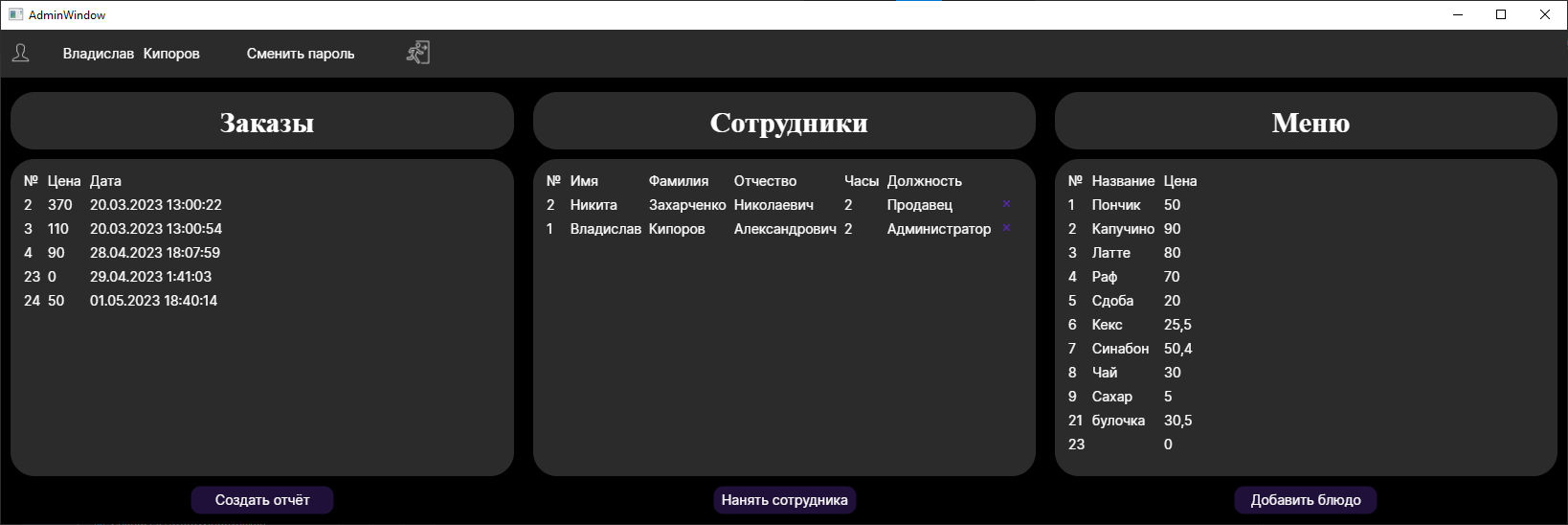


Рисунок А. 6.2 - Тест удаления пользователей. Результат

Результаты работы программы

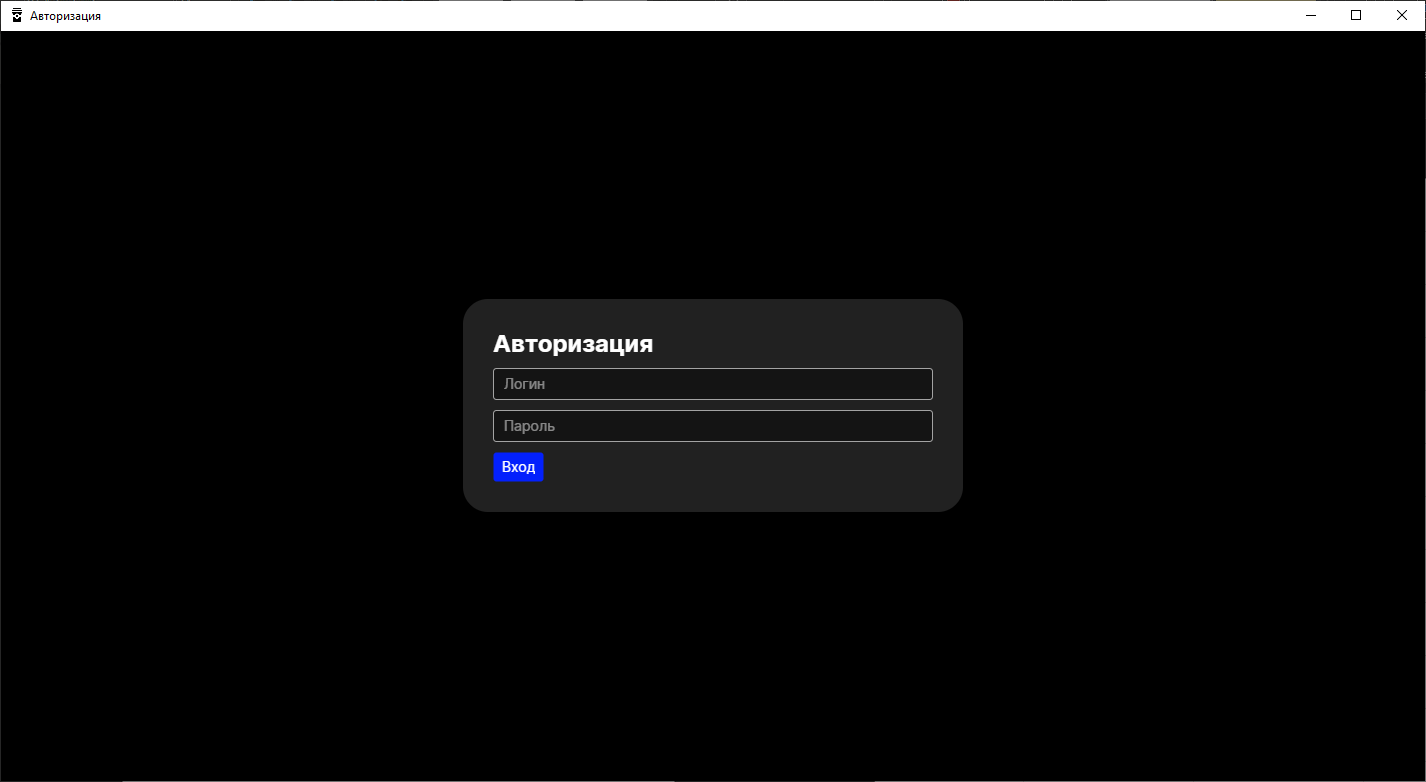


Рисунок Б. 1 – Окно авторизации

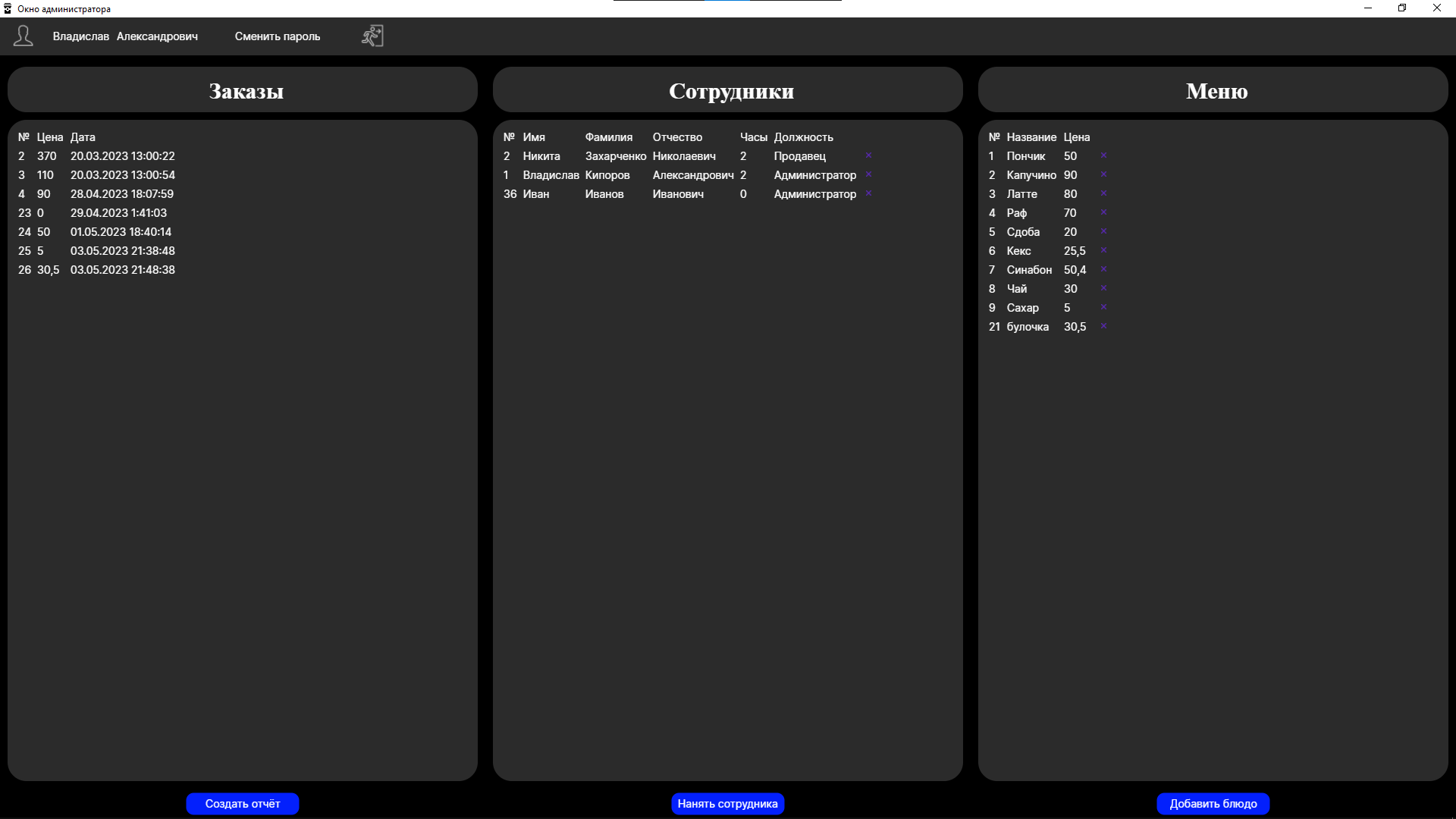


Рисунок Б. 2 – Окно администратора

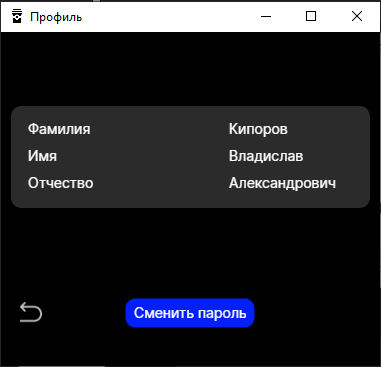


Рисунок Б. 3 – Окно профиля

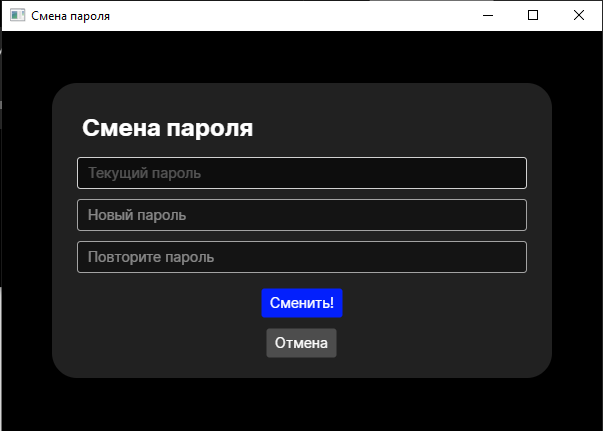


Рисунок Б. 4 - Окно смены пароля

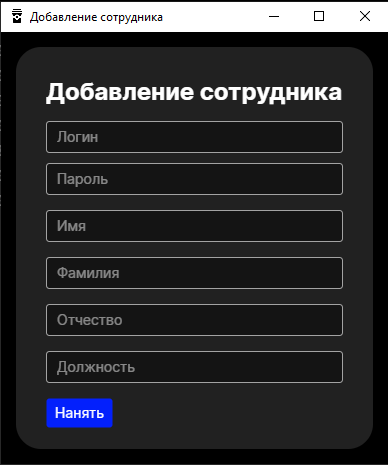


Рисунок Б. 5 - Окно регистрации пользователя

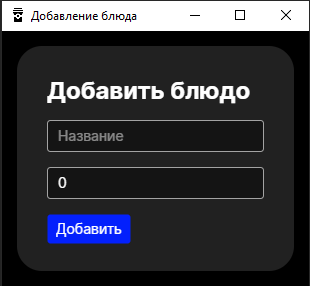


Рисунок Б. 6 - Окно добавления блюд

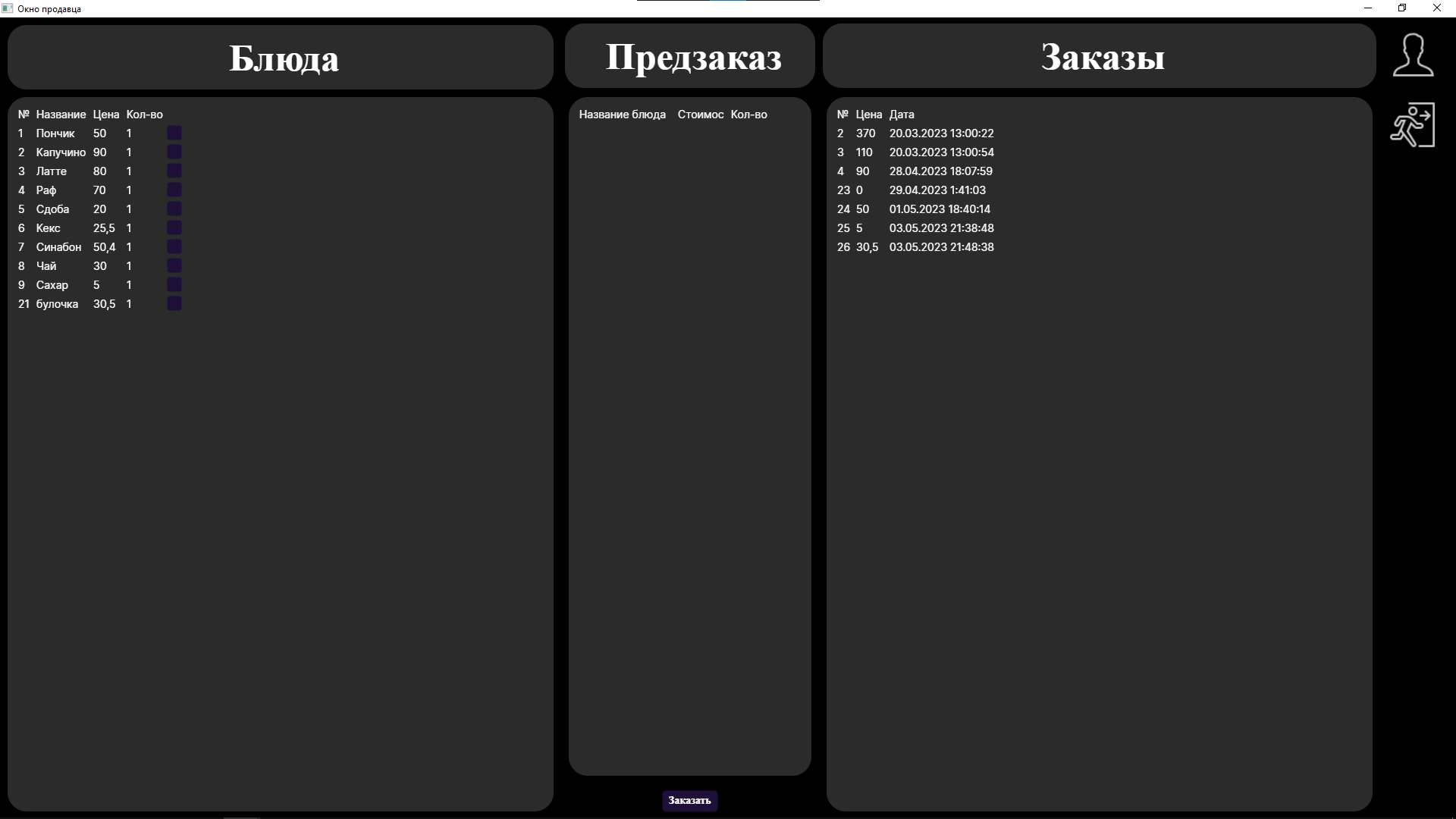


Рисунок Б. 7 - Окно продавца

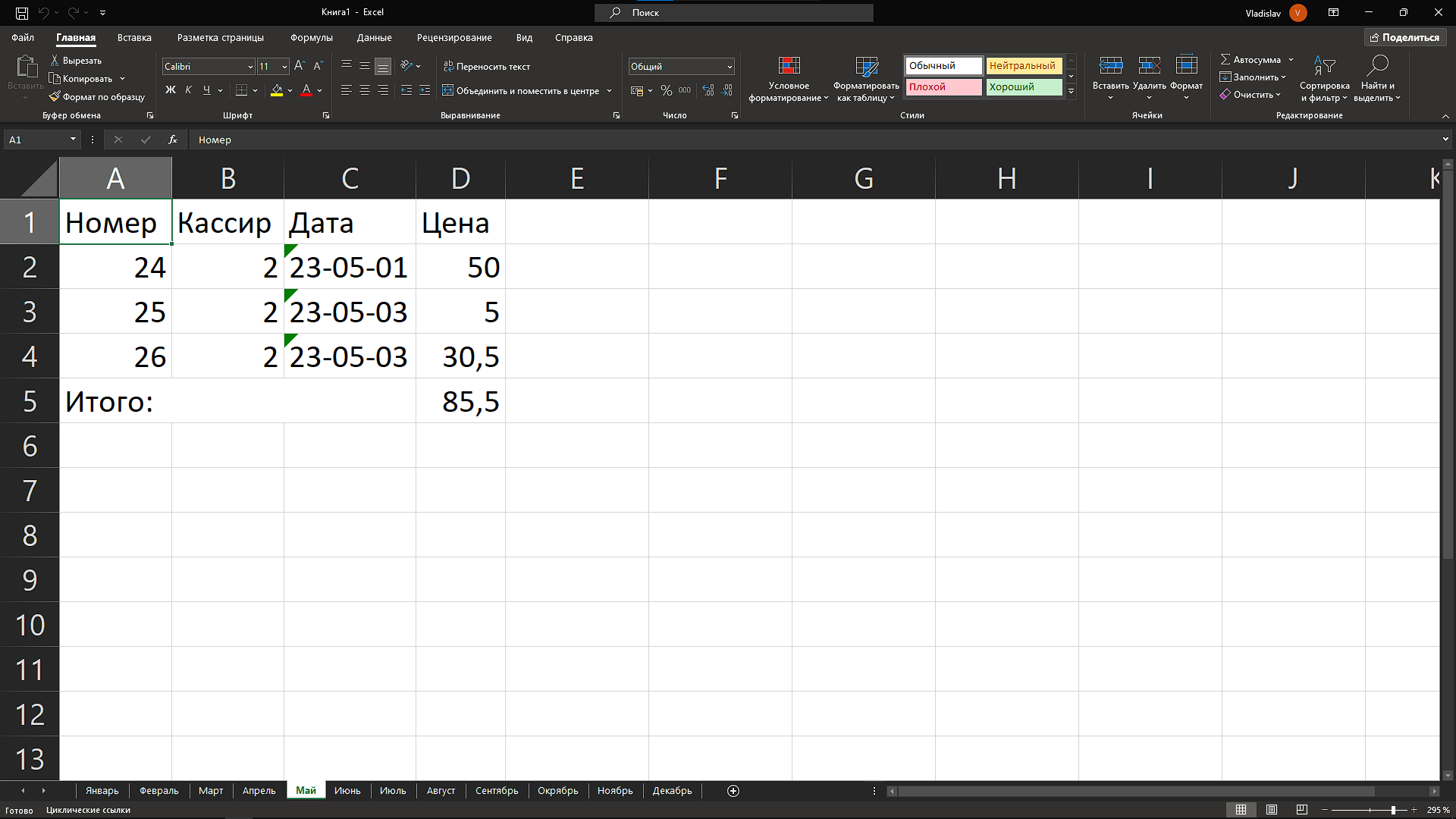


Рисунок Б. 8 – Отчёт по продажам

Листинг кода

AddDishVM.cs

using System;

using System.Linq;

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using MessageBox.Avalonia;

using MessageBox.Avalonia.Enums;

using ReactiveUI;

namespace Koffee.ViewModels;

public class AddDishVM : ViewModelBase

{

private Dish \_dish = new Dish();

private string \_name;

private float \_price;

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

public float Price

{

get => \_price;

set => this.RaiseAndSetIfChanged(ref \_price, value);

}

public ReactiveCommand<Window, Unit> AddDish { get; }

public AddDishVM()

{

AddDish = ReactiveCommand.Create<Window>(AddDishImpl);

}

private void AddDishImpl(Window obj)

{

var dish = Helper.GetContext().Dishes.FirstOrDefault(x=> x.Name == Name);

if (dish == null)

{

\_dish.Name = \_name;

\_dish.Price = \_price;

Helper.GetContext().Dishes.Add(\_dish);

try

{

Helper.GetContext().SaveChanges();

MessageBoxManager.GetMessageBoxStandardWindow("Блюдо добавлено", "Готово", ButtonEnum.Ok, Icon.Success).ShowDialog(obj);

obj.Close();

}

catch (Exception ex)

{

MessageBoxManager.GetMessageBoxStandardWindow("Данные не заполнены", "Ошибка", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

if (\_dish.Name != null & \_dish.Price != null)

{

MessageBoxManager.GetMessageBoxStandardWindow("Блюдо добавлено", "Готово", ButtonEnum.Ok, Icon.Success).ShowDialog(obj);

AuthorizationWindow authorization = new AuthorizationWindow();

}

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Данный логин занят", "Ошибка", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

}

}

AddEmployeeVM.cs

using System;

using System.Linq;

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using MessageBox.Avalonia;

using MessageBox.Avalonia.Enums;

using ReactiveUI;

namespace Koffee.ViewModels;

public class AddEmployeeVM : ViewModelBase

{

private User \_user = new User();

private string \_password;

private string \_login;

private string \_name;

private string \_surname;

private string \_patronymic;

private string \_post;

public string Login

{

get => \_login;

set => this.RaiseAndSetIfChanged(ref \_login, value);

}

public string Password

{

get => \_password;

set => this.RaiseAndSetIfChanged(ref \_password, value);

}

public string Surename

{

get => \_surname;

set => this.RaiseAndSetIfChanged(ref \_surname, value);

}

public string Name

{

get => \_name;

set => this.RaiseAndSetIfChanged(ref \_name, value);

}

public string Patronymic

{

get => \_patronymic;

set => this.RaiseAndSetIfChanged(ref \_patronymic, value);

}

public string Post

{

get => \_post;

set => this.RaiseAndSetIfChanged(ref \_post, value);

}

public ReactiveCommand<Window, Unit> AddEmployee { get; }

public AddEmployeeVM()

{

AddEmployee = ReactiveCommand.Create<Window>(AddEmployeeImpl);

}

public void AddEmployeeImpl(Window obj)

{

var user = Helper.GetContext().Users.FirstOrDefault(x=> x.Login == Login);

if (user == null)

{

\_user.Login = \_login;

\_user.Password = \_password;

\_user.Name = \_name;

\_user.Surename = \_surname;

\_user.Patronymic = \_patronymic;

\_user.Post = \_post;

\_user.HoursWorked = 0;

Helper.GetContext().Users.Add(\_user);

try

{

Helper.GetContext().SaveChanges();

MessageBoxManager.GetMessageBoxStandardWindow("Сотрудник добавлен", "Нанят", ButtonEnum.Ok, Icon.Success).ShowDialog(obj);

obj.Close();

}

catch (Exception ex)

{

MessageBoxManager.GetMessageBoxStandardWindow("Данные не заполнены", "Ошибка", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

if (\_user.Login != null & \_user.Password != null & \_user.Name != null & \_user.Surename != null & \_user.Patronymic != null & \_user.Post != null)

{

MessageBoxManager.GetMessageBoxStandardWindow("Пользователь Зарегистрирован", "Зарегистрирован", ButtonEnum.Ok, Icon.Success).ShowDialog(obj);

AuthorizationWindow authorization = new AuthorizationWindow();

}

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Данный логин занят", "Ошибка", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

}

}

AdminVM.cs

using System;

using System.Collections.ObjectModel;

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using ReactiveUI;

using System.Linq;

using System.IO;

using Koffee.Context;

using Microsoft.EntityFrameworkCore;

using Excel = Microsoft.Office.Interop.Excel;

namespace Koffee.ViewModels;

public class AdminVM : ViewModelBase

{

private ObservableCollection<Order> \_order;

private Order \_orders = new Order();

private ObservableCollection<User> \_user;

private User \_users = new User();

private static ObservableCollection<Dish> \_dishList = new ObservableCollection<Dish>();

private ObservableCollection<Dish> \_dish;

private MyDbContext db = new MyDbContext();

private static User AuthUserNow { get; set; }

public Order Orders

{

get => \_orders;

set => this.RaiseAndSetIfChanged(ref \_orders, value);

}

public ObservableCollection<Dish> DishLists

{

get => \_dishList;

set => this.RaiseAndSetIfChanged(ref \_dishList, value);

}

public ObservableCollection<Order> Order

{

get => \_order;

set => this.RaiseAndSetIfChanged(ref \_order, value);

}

public User Users

{

get => \_users;

set => this.RaiseAndSetIfChanged(ref \_users, value);

}

public ObservableCollection<User> User

{

get => \_user;

set => this.RaiseAndSetIfChanged(ref \_user, value);

}

public ObservableCollection<Dish> Dish

{

get => \_dish;

set => this.RaiseAndSetIfChanged(ref \_dish, value);

}

public ReactiveCommand<Window, Unit> ButtonProfile { get; }

public ReactiveCommand<Window, Unit> ExitProfile { get; }

public ReactiveCommand<Window, Unit> OpenWindowChangePassword { get; }

public ReactiveCommand<Window, Unit> AddEmployee { get; }

public ReactiveCommand<Window, Unit> CreateReport { get; }

public ReactiveCommand<Window, Unit> AddDish { get; }

public AdminVM()

{

AuthUserNow = AuthorizationVM.AuthUser;

ButtonProfile = ReactiveCommand.Create<Window>(OpenProfileWindow);

ExitProfile = ReactiveCommand.Create<Window>(ExitProfileImpl);

AddEmployee = ReactiveCommand.Create<Window>(AddEmployeeM);

CreateReport = ReactiveCommand.Create<Window>(CreateReportImpl);

AddDish = ReactiveCommand.Create<Window>(AddDishM);

OpenWindowChangePassword = ReactiveCommand.Create<Window>(OpenWindowChangePasswordImpl);

Order = new ObservableCollection<Order>(Helper.GetContext().Orders.Include(x => x.DishLists).ToList());

User = new ObservableCollection<User>(Helper.GetContext().Users.ToList());

Dish = new ObservableCollection<Dish>(Helper.GetContext().Dishes.ToList());

}

private void CreateReportImpl(Window obj)

{

using (ExcelHelper helper = new ExcelHelper())

{

if (helper.Open(filePath: Path.Combine(Environment.CurrentDirectory, @"D:\Документы\Отчёт о заказах.xlsx")))

{

int i = 0;

var allOrders = Order.ToList().OrderBy(p => p.Date).ToList();

var application = new Excel.Application();

string[] month = new string[12] { "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь", "Июль", "Август", "Сентябрь", "Окрябрь", "Ноябрь", "Декабрь" };

application.SheetsInNewWorkbook = month.Length;

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int j = 0; j < month.Length; ++j)

{

int counter = 0;

int startRowIndex = 1;

Excel.Worksheet worksheet = application.Worksheets.Item[j + 1];

worksheet.Name = month[j];

worksheet.Cells[1][startRowIndex] = "Номер";

worksheet.Cells[2][startRowIndex] = "Кассир";

worksheet.Cells[3][startRowIndex] = "Дата";

worksheet.Cells[4][startRowIndex] = "Цена";

startRowIndex++;

while (allOrders.Count > i)

{

if (allOrders[i].Date.Month == j + 1)

{

worksheet.Cells[1][startRowIndex] = allOrders[i].IdOrder;

worksheet.Cells[2][startRowIndex] = allOrders[i].IdUser;

worksheet.Cells[3][startRowIndex] = allOrders[i].Date.ToString("yy-MM-dd");

worksheet.Cells[4][startRowIndex] = allOrders[i].Price;

counter++;

}

else

{

break;

}

i++;

startRowIndex++;

}

Excel.Range sumRange = worksheet.Range[worksheet.Cells[1][startRowIndex],

worksheet.Cells[3][startRowIndex]];

sumRange.Merge();

sumRange.Value = "Итого:";

worksheet.Cells[4][startRowIndex].Formula =

$"=SUM(D{startRowIndex - counter}:" + $"D{startRowIndex - 1}";

worksheet.Columns.AutoFit();

helper.Save();

}

application.Visible = true;

}

}

}

private void AddDishM(Window obj)

{

AddDishWindow adm = new AddDishWindow();

adm.Show();

}

private void OpenProfileWindow(Window obj)

{

ProfileWindow profile\_window = new ProfileWindow();

profile\_window.Show();

obj.Close();

}

private void ExitProfileImpl(Window obj)

{

AuthUserNow = null;

AuthorizationWindow aw = new AuthorizationWindow();

aw.Show();

obj.Close();

}

public void RemoveEmployeeImpl(User user)

{

\_user.Remove(user);

User users = db.Users.Where(o => o.IdUser == user.IdUser).FirstOrDefault();

db.Users.Remove(users);

db.SaveChanges();

}

public void RemoveDishImpl(Dish dish)

{

\_dish.Remove(dish);

Dish dishes = db.Dishes.Where(o => o.IdDish == dish.IdDish).FirstOrDefault();

db.Dishes.Remove(dishes);

db.SaveChanges();

}

private void OpenWindowChangePasswordImpl(Window obj)

{

ChangePasswordWindow changePass = new ChangePasswordWindow();

changePass.Show();

}

private void AddEmployeeM(Window obj)

{

AddEmployeeWindow aem = new AddEmployeeWindow();

aem.Show();

}

}

AuthorizationVM.cs

using System.Linq;

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using MessageBox.Avalonia;

using MessageBox.Avalonia.Enums;

using ReactiveUI;

namespace Koffee.ViewModels;

public class AuthorizationVM : ViewModelBase

{

private string \_password;

private string \_login;

public static User AuthUser { get; set; }

private User \_user;

public User User

{

get => \_user;

set => this.RaiseAndSetIfChanged(ref \_user, value);

}

public string Login

{

get => \_login;

set => this.RaiseAndSetIfChanged(ref \_login, value);

}

public string Password

{

get => \_password;

set => this.RaiseAndSetIfChanged(ref \_password, value);

}

public ReactiveCommand<Window, Unit> ButtonEnter { get; }

public AuthorizationVM()

{

ButtonEnter = ReactiveCommand.Create<Window>(OpenWindowImpl);

}

private void OpenWindowImpl(Window obj)

{

User user = null;

user = Helper.GetContext().Users.SingleOrDefault(x => x.Login == Login & x.Password == Password);

if (user != null)

{

SingIn(user, obj);

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Ошибка", "Логин или пароль неверны!",

ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

}

}

public void OpenWindowImp(string Login, string Password)

{

User user = null;

user = Helper.GetContext().Users.SingleOrDefault(x => x.Login == Login & x.Password == Password);

}

private void SingIn(User user, Window obj)

{

AuthUser = user;

if (user.Post == "Продавец")

{

SellerWindow homePage = new SellerWindow();

homePage.Show();

obj.Close();

}

else if(user.Post == "Администратор")

{

AdminWindow homePage = new AdminWindow();

homePage.Show();

obj.Close();

}

}

}

ChangePasswordVm.cs

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using MessageBox.Avalonia;

using MessageBox.Avalonia.Enums;

using ReactiveUI;

namespace Koffee.ViewModels;

public class ChangePasswordVM : ViewModelBase

{

private string \_firstpassword;

private string \_secondpassword;

private string \_oldpassword;

public static User AuthUser { get; set; }

public string OldPassword

{

get => \_oldpassword;

set => this.RaiseAndSetIfChanged(ref \_oldpassword, value);

}

public string FirstPassword

{

get => \_firstpassword;

set => this.RaiseAndSetIfChanged(ref \_firstpassword, value);

}

public string SecondPassword

{

get => \_secondpassword;

set => this.RaiseAndSetIfChanged(ref \_secondpassword, value);

}

public ReactiveCommand<Window, Unit> ChangeButton { get; }

public ReactiveCommand<Window, Unit> Cancel { get; }

public ChangePasswordVM()

{

ChangeButton = ReactiveCommand.Create<Window>(SubmitChangeImpl);

Cancel = ReactiveCommand.Create<Window>(CancelImpl);

}

private void CancelImpl(Window obj)

{

obj.Close();

}

private void SubmitChangeImpl(Window obj)

{

if (\_oldpassword == AuthorizationVM.AuthUser.Password)

{

if (\_oldpassword != \_firstpassword)

{

if (\_firstpassword == \_secondpassword)

{

AuthorizationVM.AuthUser.Password = \_firstpassword;

Helper.GetContext().Users.Update(AuthorizationVM.AuthUser);

Helper.GetContext().SaveChanges();

MessageBoxManager.GetMessageBoxStandardWindow("Успешно","Пароль изменён", ButtonEnum.Ok, Icon.Success).ShowDialog(obj);

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Ошибка","Пароли не совпадают", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Ошибка","Нельзя использовать старый пароль", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

}

else

{

MessageBoxManager.GetMessageBoxStandardWindow("Ошибка","Неверный текущий пароль", ButtonEnum.Ok, Icon.Error).ShowDialog(obj);

return;

}

}

}

ProfileVM.cs

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using ReactiveUI;

namespace Koffee.ViewModels;

public class ProfileVM : ViewModelBase

{

private static User AuthUserNow { get; set; }

private User \_user;

public ReactiveCommand<Window, Unit> GoBack { get; }

public ReactiveCommand<Window, Unit> OpenWindowChangePassword { get; }

public ProfileVM()

{

AuthUserNow = AuthorizationVM.AuthUser;

GoBack = ReactiveCommand.Create<Window>(GoBackImpl);

OpenWindowChangePassword = ReactiveCommand.Create<Window>(OpenWindowChangePasswordImpl);

}

private void OpenWindowChangePasswordImpl(Window obj)

{

ChangePasswordWindow changePass = new ChangePasswordWindow();

changePass.Show();

}

private void GoBackImpl(Window obj)

{

if (AuthUserNow.Post == "Продавец")

{

SellerWindow sellerWindow = new SellerWindow();

sellerWindow.Show();

obj.Close();

}

else

{

AdminWindow adminWindow = new AdminWindow();

adminWindow.Show();

obj.Close();

}

}

}

SellerVM.cs

using System;

using System.Collections.ObjectModel;

using System.Reactive;

using Avalonia.Controls;

using Koffee.Models;

using Koffee.Views;

using ReactiveUI;

using System.Linq;

using AvaloniaEdit.Utils;

using MessageBox.Avalonia;

using MessageBox.Avalonia.Enums;

using Microsoft.EntityFrameworkCore;

namespace Koffee.ViewModels;

public class SellerVM : ViewModelBase

{

private ObservableCollection<Dish> \_dish;

private ObservableCollection<Order> \_order;

private Order \_orders = new Order();

private static ObservableCollection<Dish> \_dishList = new ObservableCollection<Dish>();

private ObservableCollection<DishList> \_selectedDishs = new ObservableCollection<DishList>();

private ObservableCollection<Dish> \_dishInPreorder = new ObservableCollection<Dish>();

public static ObservableCollection<Dish> Dishes => \_dishList;

private ObservableCollection<Dish> \_ordering = new ObservableCollection<Dish>();

private static User AuthUserNow { get; set; }

public Order Orders

{

get => \_orders;

set => this.RaiseAndSetIfChanged(ref \_orders, value);

}

public ObservableCollection<DishList> SelectedDishs

{

get => \_selectedDishs;

set => this.RaiseAndSetIfChanged(ref \_selectedDishs, value);

}

public ObservableCollection<Dish> Dish

{

get => \_dish;

set => this.RaiseAndSetIfChanged(ref \_dish, value);

}

public ObservableCollection<Order> Order

{

get => \_order;

set => this.RaiseAndSetIfChanged(ref \_order, value);

}

public ObservableCollection<Dish> DishLists

{

get => \_dishList;

set => this.RaiseAndSetIfChanged(ref \_dishList, value);

}

public ObservableCollection<Dish> DishInPreorder

{

get => \_dishInPreorder;

set => this.RaiseAndSetIfChanged(ref \_dishInPreorder, value);

}

public ReactiveCommand<Window, Unit> ButtonProfile { get; }

public ReactiveCommand<Window, Unit> SubmitOrder { get; }

public ReactiveCommand<Window, Unit> ExitProfile { get; }

public SellerVM()

{

AuthUserNow = AuthorizationVM.AuthUser;

ButtonProfile = ReactiveCommand.Create<Window>(OpenProfileWindow);

ExitProfile = ReactiveCommand.Create<Window>(ExitProfileImpl);

SubmitOrder = ReactiveCommand.Create<Window>(SubmitOrderImpl);

Dish = new ObservableCollection<Dish>(Helper.GetContext().Dishes.ToList());

Order = new ObservableCollection<Order>(Helper.GetContext().Orders.Include(x => x.DishLists).ThenInclude(x => x.IdDishNavigation).ToList());

}

private void SubmitOrderImpl(Window obj)

{

CreateOrder(obj);

}

private void OpenProfileWindow(Window obj)

{

ProfileWindow profile\_window = new ProfileWindow();

profile\_window.Show();

obj.Close();

}

private void ExitProfileImpl(Window obj)

{

AuthUserNow = null;

AuthorizationWindow aw = new AuthorizationWindow();

aw.Show();

obj.Close();

}

private void CreateOrder(Window obj)

{

var context = Helper.GetContext();

var dishes = context.Dishes

.Where(x => \_dishList.Select(x => x.IdDish).Contains(x.IdDish))

.ToList();

Order order = new Order();

order.Date = DateTime.Now;

order.IdUser = AuthUserNow.IdUser;

order.DishLists = dishes.Select(x => new DishList { IdDishNavigation = x, Count = x.Count }).ToList();

order.Price = \_dishList.Sum(x => x.Price);

Helper.GetContext().Orders.Add(order);

Helper.GetContext().Orders.UpdateRange();

Helper.GetContext().SaveChanges();

\_order.Add(order);

\_dishList.Clear();

}

private void DishChecking()

{

ProfileWindow profile\_window = new ProfileWindow();

profile\_window.Show();

}

public void AddDishPreorderImpl(Dish dish)

{

var edentity = \_dishList.SingleOrDefault(x => x.Name == dish.Name);

if (edentity == null)

{

\_dishList.Add(new Dish

{

IdDish = dish.IdDish,

Name = dish.Name,

Price = dish.Price \* dish.Count,

Count = dish.Count

});

}

else

{

\_dishList.Remove(edentity);

\_dishList.Add(new Dish

{

IdDish = dish.IdDish,

Name = dish.Name,

Price = dish.Price \* dish.Count,

Count = dish.Count

});

}

}

public void RemoveDishPreorderImpl(Dish dish)

{

\_dishList.Remove(dish);

}

}

ExcelHelper.cs

using System;

using System.IO;

using Excel = Microsoft.Office.Interop.Excel;

namespace Koffee;

class ExcelHelper : IDisposable

{

private Excel.Application \_excel;

private Excel.Workbook \_workbook;

private string \_filePath;

public ExcelHelper()

{

\_excel = new Excel.Application();

}

public void Dispose()

{

try

{

\_workbook.Close();

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

internal bool Open(string filePath)

{

try

{

if (File.Exists(filePath))

{

\_workbook = \_excel.Workbooks.Open(filePath);

}

else

{

\_workbook = \_excel.Workbooks.Add();

\_filePath = filePath;

}

return true;

}

catch (Exception e)

{

return false;

}

}

internal bool Set(string column, int row, object data)

{

try

{

((Excel.Worksheet) \_excel.ActiveSheet).Cells[row, column] = data;

return true;

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

public void Save()

{

if (!string.IsNullOrEmpty( \_filePath))

{

\_workbook.SaveAs(\_filePath);

\_filePath = null;

}

else

{

\_workbook.Save();

}

}

}

Helper.cs

using Koffee.Context;

namespace Koffee;

public class Helper

{

private static MyDbContext \_satellitecontext;

public static MyDbContext GetContext()

{

return \_satellitecontext ??= new();

}

}

Program.cs

using Avalonia;

using Avalonia.ReactiveUI;

using System;

using Avalonia.Controls;

using Koffee.Context;

using Koffee.Views;

namespace Koffee;

class Program

{

[STAThread]

public static void Main(string[] args) => BuildAvaloniaApp()

.StartWithClassicDesktopLifetime(args);

public static AppBuilder BuildAvaloniaApp()

=> AppBuilder.Configure<App>()

.UsePlatformDetect()

.LogToTrace()

.UseReactiveUI();

private static void AppMain(Application app, string[] args)

{

var window = new AuthorizationWindow()

{

DataContext = new MyDbContext(),

};

app.Run(window);

}

}