**МИНОБРНАУКИ РОССИИ**

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ОТЧЕТ**

**по учебной практике УП.02.01 (по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность 09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент 4 курса 42919/9 группы

Михеев Александр Сергеевич

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, пр. Энгельса, 23

(наименование и адрес организации)

Период прохождения практики

с «09» сентября 2024 г. по «21» сентября 2024 г.

Руководитель практики Хисамутдинова А.С.

(подпись) (расшифровка подписи)

Итоговая оценка по практике

Санкт-Петербург 2024

**ЗАДАНИЕ**

**на учебную практику (по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность 09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент 4 курса 42919/9 группы

Михеев Александр Сергеевич

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, пр. Энгельса, 23

(наименование и адрес организации)

Период прохождения практики

с «09» сентября 2024 г. по «21» сентября 2024 г.

**Виды работ, обязательные для выполнения** *(переносится из программы, соответствующего ПМ):*

* Участие в выработке требований к программному обеспечению;
* Стадии проектирования программного обеспечения;
* Разработка модулей программного обеспечения;
* Тестирование программных модулей и их интеграции;
* Разработка программной документации и стандарты кодирования.

**Индивидуальное задание: ВАРИАНТ 10**

Задание выдал «09» сентября 2024 г. Хисамутдинова А.С.

(подпись) (Ф.И.О.)

Задание получил «09» сентября 2024 г. Михеев А.С.

(подпись) (Ф.И.О.)

**МИНОБРНАУКИ РОССИИ**

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ДНЕВНИК**

**прохождения учебной практики УП.02.01 (по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность 09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент 4 курса 42919/9 группы

Михеев Александр Сергеевич

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, пр. Энгельса, 23

(наименование и адрес организации)

Период прохождения практики

с «09» сентября 2024 г. по «21» сентября 2024 г.

Руководитель практики Хисамутдинова А.С.

(подпись) (расшифровка подписи)

Итоговая оценка по практике

Санкт-Петербург 2024

**Содержание дневника**

|  |  |  |
| --- | --- | --- |
| **Дата** | **Виды выполненных работ и заданий по программе практики** | **Подпись**  **руководителя практики** |
| **1** | **2** | **3** |
| 09.09.2024 | Анализ предметной области. Выявление требований к программе. |  |
| 10.09.2024 | Разработка технического задания. UML. Проектирование диаграммы вариантов использования. |  |
| 11.09.2024 | UML. Проектирование диаграммы последовательности. UML. Проектирование диаграммы активности. |  |
| 12.09.2024 | Моделирование структуры ПО. Проектирование инфологической и даталогической модели данных. |  |
| 13.09.2024 | Проектирование интерфейса пользователя. Создание Wireframe эскизов. Разработка дизайна программы в соответствии с руководством по стилю. |  |
| 14.09.2024 | Разработка базы данных. Разработка словаря данных. |  |
| 16.09.2024 | Создание приложения. Форма авторизации. Создание приложения. Форма заказов. |  |
| 17.09.2024 | Создание приложения. Основные формы приложения. Разработка библиотеки классов. |  |
| 18.09.2024 | Подготовка отчетов и выгрузка документов для печати. Отладка программных модулей. |  |
| 19.09.2024 | Модульное тестирование. Создание тестовых случаев. |  |
| 20.09.2024 | Интеграционное тестирование. Разработка самодокументирующегося кода. |  |
| 21.09.2024 | Инспекция кода на соответствие стандартам кодирования. Подготовка отчета и размещение результатов в репозитории контроля версий. |  |

СОДЕРЖАНИЕ

[Введение 6](#_Toc178313991)

[Задание 1. Разработка технического задания 7](#_Toc178313992)

[Описание предметной области 7](#_Toc178313993)

[Техническое задание 9](#_Toc178313994)

[Задание 2. Разработка алгоритмов и диаграмм 13](#_Toc178313995)

[База данных 13](#_Toc178313996)

[Диаграммы 21](#_Toc178313997)

[Задание 3. Разработка макетов программы 25](#_Toc178313998)

[Задание 4. Разработка программы 34](#_Toc178313999)

[Реализация приложения 34](#_Toc178314000)

[Отладка приложения 51](#_Toc178314001)

[Задание 5. Тестирование 53](#_Toc178314002)

[Тестирование 53](#_Toc178314003)

[Unit-тесты 63](#_Toc178314004)

[Задание 6. GitHub. Оценка проекта 64](#_Toc178314005)

[ПРИЛОЖЕНИЕ А 65](#_Toc178314006)

[SQL-запросы 65](#_Toc178314007)

[Код приложения 73](#_Toc178314008)

[unit-тесты 94](#_Toc178314009)

# Введение

Фабрика по пошиву одежды "Модный пошив" занимается производством и реализацией высококачественной одежды различных моделей и размеров. Для повышения эффективности работы и оптимизации производственных процессов фабрика решила автоматизировать учет, планирование и управление производством с помощью специально разработанного программного продукта.

Основной целью данного программного продукта является автоматизация процессов учета, планирования и управления производством на фабрике "Модный пошив". Программный продукт позволяет сотрудникам фабрики прозрачно и эффективно управлять заказами, отслеживать ход производства, контролировать запасы материалов и готовой продукции.

Задача разработки такого программного продукта состоит в создании удобного и эффективного инструмента для:

* Учета и контроля заказов от клиентов.
* Планирования и управления производством.
* Контроля запасов материалов и готовой продукции.
* Генерации отчетов и анализа производственных показателей.

Программный продукт направлен на повышение эффективности работы фабрики, оптимизацию производственных процессов и улучшение качества обслуживания клиентов.

# Задание 1. Разработка технического задания

## Описание предметной области

**Основной целью** данного программного продукта является автоматизация процессов учета, планирования и управления производством на фабрике по пошиву одежды «Модный пошив». Программный продукт позволяет сотрудникам фабрики прозрачно и эффективно управлять заказами, отслеживать ход производства, контролировать запасы материалов и готовой продукции.

**Разработка программного продукта** для фабрики по пошиву одежды — это процесс создания программного решения, которое позволяет автоматизировать и оптимизировать все этапы производственного цикла, начиная от приема заказов и заканчивая отгрузкой готовой продукции клиентам.

Программный продукт направлен на повышение эффективности работы фабрики «Модный пошив», оптимизацию производственных процессов и улучшение качества обслуживания клиентов.

**Задача разработки** такого программного продукта состоит в создании удобного и эффективного инструмента для:

* Учета и контроля заказов от клиентов.
* Планирования и управления производством.
* Контроля запасов материалов и готовой продукции.
* Генерации отчетов и анализа производственных показателей.

**Программный продукт** позволяет фабрике «Модный пошив» повысить эффективность и точность учета, сократить время обработки заказов, оптимизировать использование ресурсов и повысить удовлетворенность клиентов.

**Основные функции и возможности программного продукта включают:**

1. **Управление заказами:**

* **Прием и регистрация заказов:** Операторы фабрики регистрируют заказы от клиентов, указывая тип одежды, модель, размеры, количество и сроки выполнения.
* **Отслеживание статуса заказов:** клиенты и сотрудники фабрики могут отслеживать статус заказов (новый, в работе, готов, отгружен).
* **Редактирование и удаление заказов:** при необходимости можно изменить или удалить заказ.

**2. Управление производством:**

* **Планирование производства:** на основе заказов создаются производственные задания, назначаются исполнители (швеи, закройщики) и устанавливаются сроки выполнения.
* **Отслеживание хода производства:** слежение за выполнением производственных заданий, контроль сроков и качества работ.
* **Учет материалов:** регистрация поступления материалов на склад, отслеживание остатков, формирование заявок на закупку.
* **Учет готовой продукции:** Регистрация готовой продукции, контроль остатков на складе.

**3. Администрирование:**

* **Управление пользователями:** добавление, редактирование и удаление пользователей, назначение ролей (администратор, менеджер по продажам, мастер цеха, кладовщик).
* **Управление доступом:** настройка прав доступа для каждой роли.
* **Генерация отчетов:** формирование отчетов по заказам, производству, материалам и готовой продукции.

**4. Интерфейс и пользовательский опыт:**

* **Простой и интуитивный интерфейс:** удобный и понятный интерфейс для всех категорий пользователей.
* **Информативные уведомления и подсказки:** предоставление пользователям актуальной информации и помощи в работе с программным продуктом.

5. Отчётность и информирование:

* Важной составляющей учёта заявок на ремонт является фиксация и отчёт о выполненной работе. После производства продукта должен предоставить отчёт о проделанной работе, включая информацию о затраченных ресурсах (время, материалы) и оказанной помощи.

6. Мониторинг и анализ:

* Этот этап предполагает контроль и анализ процесса учёта заявок на пошив одежды. Важно отслеживать и анализировать время обработки заявок, качество выполненных работ, расходы и прочие параметры, которые могут помочь в оптимизации и улучшении процесса.

## Техническое задание

1. **Общие сведения**
   1. **Наименование проекта:** разработка программного модуля для учета заявок на фабрике по пошиву одежды.
   2. **Заказчик:** Склад «СкладЭкспресс».
   3. **Исполнитель:** Фабрика одежды «Модный пошив».
2. **Функциональные требования**
   1. **Управление заказами**
      1. **Добавление заказа:**

* Номер заказа.
* Дата заказа.
* Клиент (ФИО, контактная информация).
* Тип одежды (мужская, женская, детская).
* Модель одежды.
* Размеры.
* Количество.
* Сроки выполнения.
* Статус заказа (новый, в работе, готов, отгружен).
  + 1. **Редактирование заказа:**
* Изменение любого параметра заказа.
* Изменение статуса заказа.
  + 1. **Отслеживание статуса заказа:**
* Отображение списка заказов с фильтрацией по статусу, клиенту, срокам.
* Поиск заказа по номеру или параметрам.
  + 1. **Удаление заказа:**
* Возможность удаления заказа из системы.
  1. **Управление производством**
     1. **Планирование производства:**
* Создание производственных заданий на основе заказов.
* Установка сроков выполнения заданий.
  + 1. **Отслеживание хода производства:**
* Отображение списка заданий с фильтрацией по статусу, исполнителю, срокам.
* Поиск задания по номеру или параметрам.
  + 1. **Учет материалов:**
* Регистрация поступления материалов на склад.
  + 1. **Учет готовой продукции:**
* Регистрация готовой продукции.
* Отслеживание остатков готовой продукции на складе.
  1. **Администрирование**
     1. **Управление пользователями:**
* Добавление, редактирование, удаление пользователей.
* Назначение ролей пользователям (администратор, менеджер по продажам, мастер цеха, кладовщик).
  + 1. **Управление доступом:**
* Настройка прав доступа для каждой роли.
  + 1. **Генерация отчетов:**
* Отчеты по заказам (выполненные, просроченные, в работе).
* Отчеты по производству (выполненные задания, простои).
* Отчеты по материалам (расход, остатки).
* Отчеты по готовой продукции (произведено, отгружено).

1. **Нефункциональные требования**
   1. **Кроссплатформенность:**

* Поддержка работы на ОС семейства Windows.
  1. **Безопасность:**
* Логин и пароль для доступа к приложению.
* Доступ к данным должен быть ограничен в зависимости от роли пользователя.
  1. **Удобство использования:**
* Простой и интуитивный интерфейс.
* Информативные уведомления и подсказки.
  1. **Производительность:**
* Приложение должно иметь быстрый доступ к данным.
* Минимальное время отклика на запросы пользователя.

1. **Требования к реализации**
   1. **Язык программирования:**

* Python 3.12
  1. **СУБД:**
* PostgreSQL

1. **Требования к документации**
   1. **Техническое задание на разработку программного продукта.**
2. **Руководство по стилю**
   1. **Шрифт:**

* Arial, 12 пт.
  1. **Цветовая схема:**
* Фон: #bbbbbb
* Текст: #293133
* Кнопки: #a2a2d0.

# Задание 2. Разработка алгоритмов и диаграмм

## База данных

ER-модель БД для фабрики по производству одежды «Модный пошив» (рисунок 1).

Изображение выглядит как снимок экрана, Графическое программное обеспечение, Мультимедийное программное обеспечение, 3D-моделирование

Автоматически созданное описание  
Рисунок 1 — ER-модель БД

Словарь данных для базы данных фабрики по пошиву одежды "Модный пошив".

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип данных** | **Ограничения** | **Описание** |
| Таблица «Order» (Заказы) | | | |
| order\_id | int | Primary Key | Уникальный идентификатор заказа. |
| order\_date | timestamp | Not Null | Дата создания заказа. |
| user\_id | int | Not Null | Идентификатор пользователя, создавшего заказ. |
| period\_of\_execution | timestamp | Not Null | Срок выполнения заказа. |
| order\_status | varchar(20) | Not Null | Статус заказа |
| Таблица «OrderLine» (Позиция заказа) | | | |
| order\_line\_id | int | Primary Key | Уникальный идентификатор позиции заказа. |
| order\_id | int | Not Null | Идентификатор заказа, к которому относится позиция. |
| product\_id | int | Not Null | Идентификатор продукта в позиции. |
| quantity | int | Default: 1 | Количество продукта в позиции (по умолчанию 1). |
| total\_price | decimal(10, 2) | Not Null | Итоговая цена с учётом количества продукта. |
| Таблица «Material» (Материалы) | | | |
| material\_id | int | Primary Key | Уникальный идентификатор материала. |
| material\_name | varchar(255) | Not Null | Название материала. |
| Таблица «User» (Пользователи) | | | |
| user\_id | int | Primary Key | Уникальный идентификатор пользователя. |
| role\_id | int | Not Null | Идентификатор роли пользователя. |
| full\_name | text | Not Null | Полное имя пользователя. |
| login | text | Not Null, Unique | Логин пользователя (уникальный). |
| password | text | Not Null | Пароль пользователя. |
| phone | varchar(20) |  | Телефон пользователя. |
| Таблица «Product» (Товары) | | | |
| product\_id | int | Primary Key | Уникальный идентификатор продукта. |
| type\_id | int | Not Null | Идентификатор типа продукта. |
| model\_id | int | Not Null | Идентификатор модели продукта. |
| size | varchar(10) | Not Null | Размер продукта. |
| color | varchar(20) | Not Null | Цвет продукта. |
| material\_id | int | Not Null | Идентификатор материала, из которого изготовлен продукт. |
| price | decimal(10, 2) | Not Null | Цена продукта. |
| Таблица «ProductModel» (Модель товара) | | | |
| model\_id | int | Primary Key | Уникальный идентификатор модели. |
| model\_name | varchar(255) | Not Null, Unique | Название модели (уникальное). |
| Таблица «ProductType» (Тип товара) | | | |
| type\_id | int | Primary Key | Уникальный идентификатор типа. |
| type\_name | varchar(255) | Not Null, Unique | Название типа (уникальное). |
| Таблица «ProductionTarget» (Производственное задание) | | | |
| target\_id | int | Primary Key | Уникальный идентификатор производственного задания. |
| order\_id | int | Not Null | Идентификатор заказа, для которого создано задание. |
| performer\_id | int | Not Null | Идентификатор исполнителя задания. |
| period\_of\_execution | timestamp | Not Null | Срок выполнения задания. |
| target\_status | varchar(20) | Not Null | Статус задания (например, "в работе", "выполнено"). |
| Таблица «Warehouse» (Склад) | | | |
| product\_id | int | Primary Key | Уникальный идентификатор продукта на складе. |
| quantity | int | Default: 0 | Количество продукта на складе (по умолчанию 0). |
| Таблица «Role» (Роли) | | | |
| role\_id | int | Primary Key | Уникальный идентификатор роли. |
| role\_name | varchar(255 | Not Null, Unique | Название роли (уникальное). |

Заполнение таблиц в БД PostgreSQL (рисунки 2–10).

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 1 — таблица «Users»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2 — таблица «Materials»

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 3 — таблица «OrderLines»

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 4 — таблица «Orders»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 5 — таблица «ProductionTargets»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 6 — таблица «ProductModels»

Изображение выглядит как снимок экрана, текст, число

Автоматически созданное описание

Рисунок 7 — таблица «Product»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 8 — таблица «ProductTypes»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 9 — таблица «Roles»

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 10 — таблица «Warehouse»

## Диаграммы

Диаграмма вариантов использования (прецедентов) (рисунок 11).

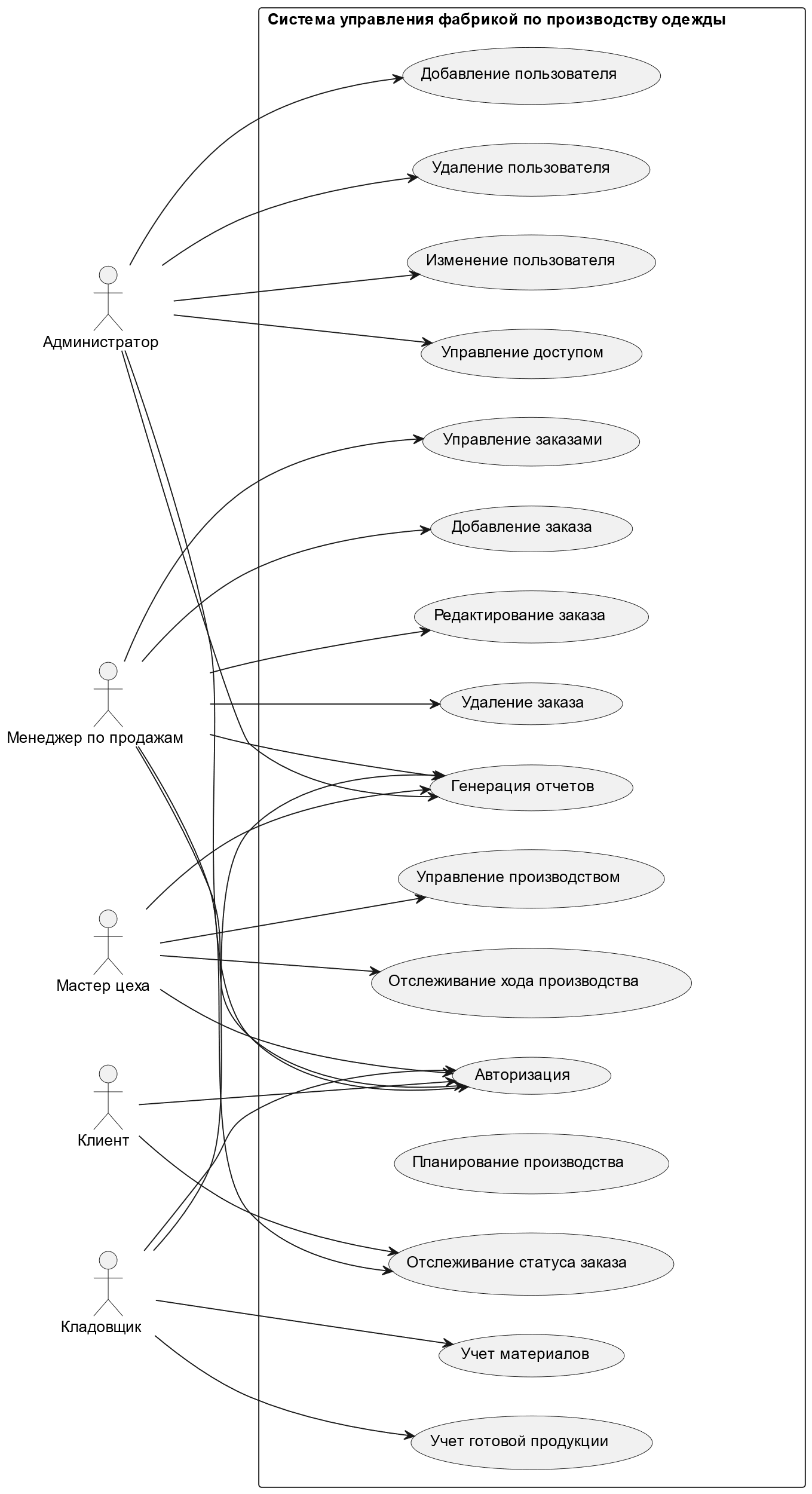
  
Рисунок 11 — Диаграмма

Диаграмма последовательности (рисунок 12).

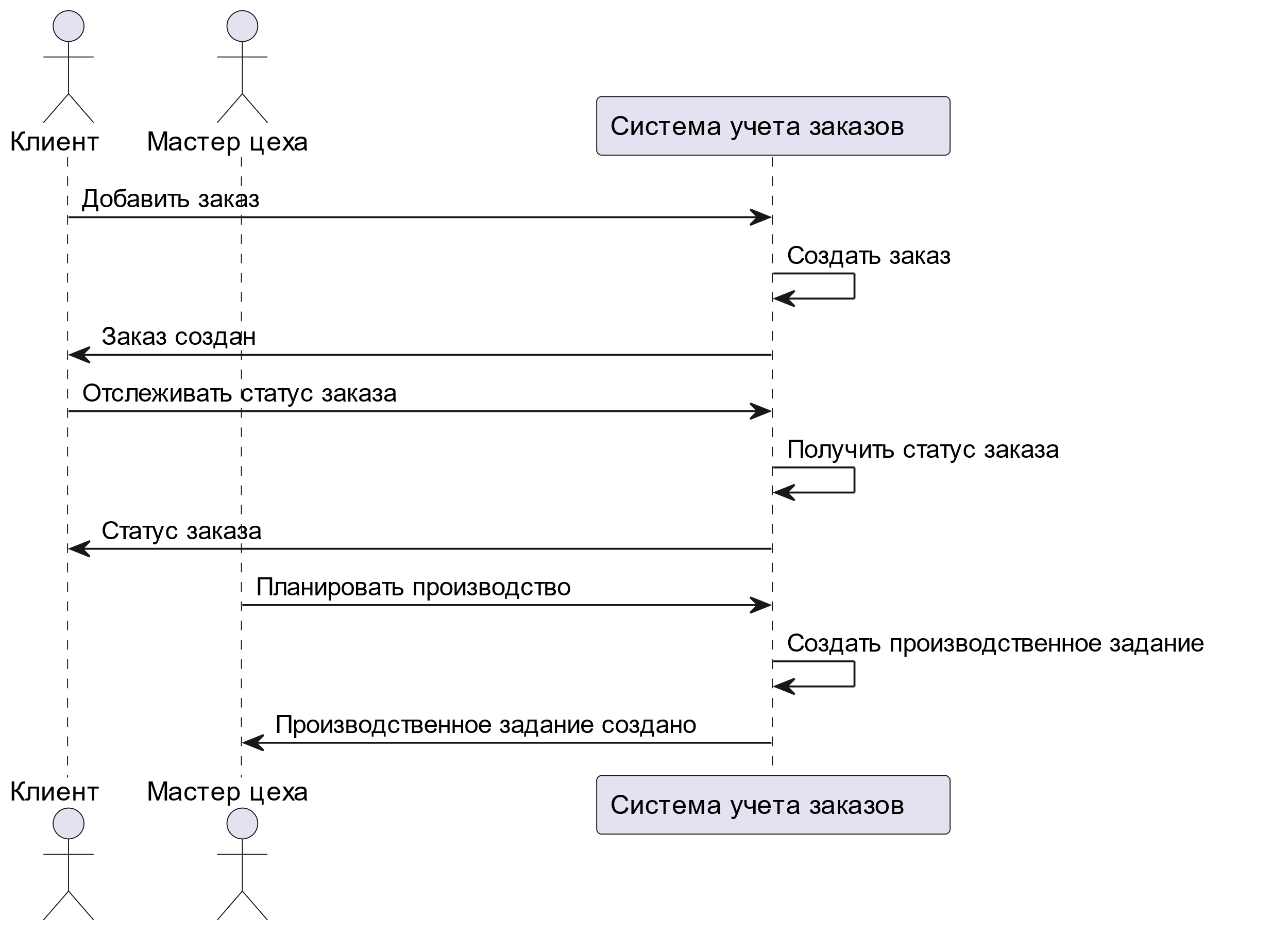
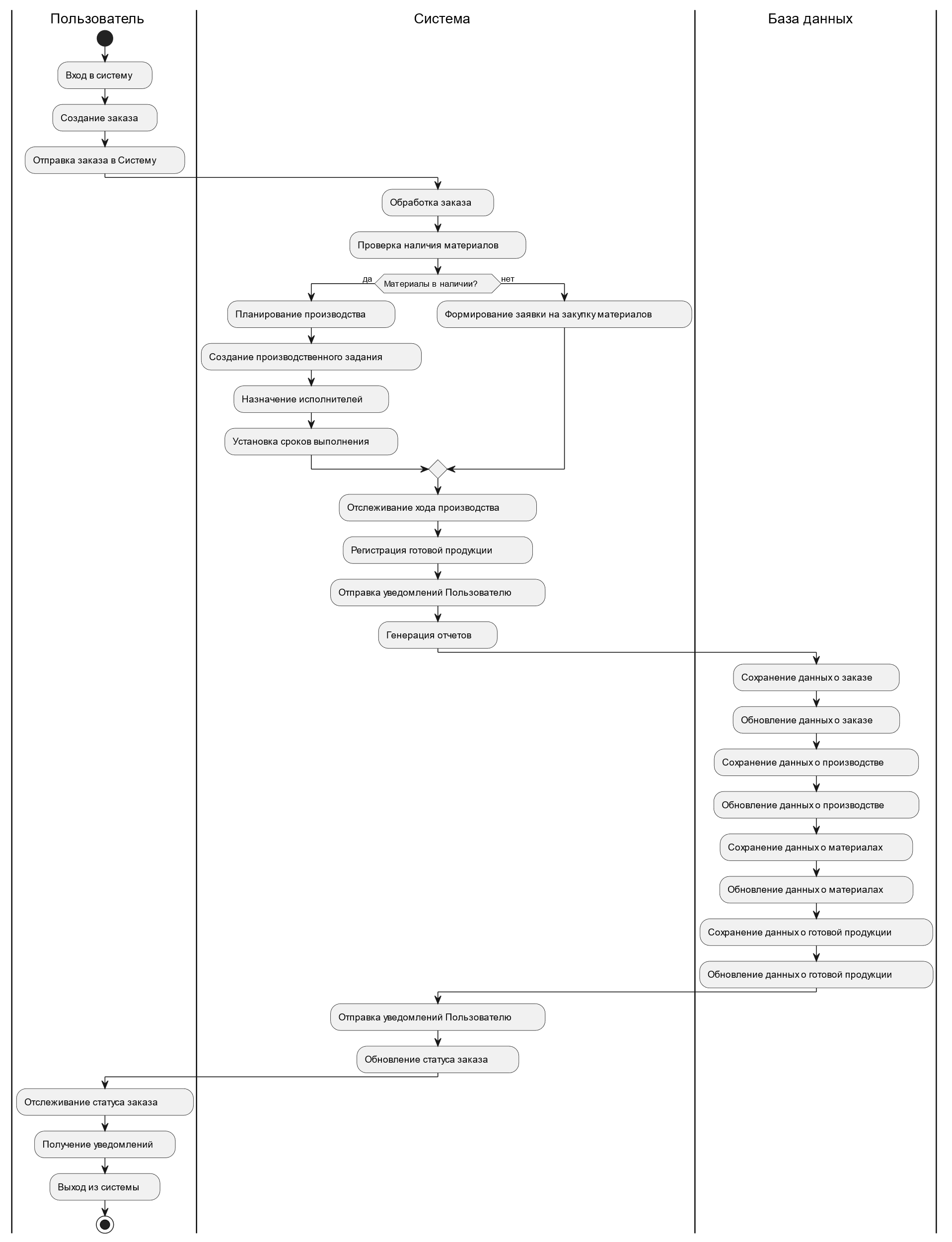
  
Рисунок 12 — Диаграмма

Диаграмма активности (рисунок 13).

  
Рисунок 13 — Диаграмма

# Задание 3. Разработка макетов программы

Были разработаны Wireframe (черно-белые) и Mockup (цветные) макеты и карта навигации приложения для фабрики по производству одежды «Модный Пошив» (рисунки 14–30).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 14 — Черно-белое окно авторизации

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 15 — Черно-белое окно «Главная»

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 16 — Черно-белое окно «Заказы»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 17 — Черно-белое окно «Производство»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 18 — Черно-белая форма редактирования заказа

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 19 — Черно-белое окно «Администрирование»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 20 — Черно-белая форма редактирования профиля пользователя

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 21 — Черно-белое окно «Отчёты»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 22 — Цветное окно авторизации

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 23 — Цветное окно «Главная»

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 24 — Цветное окно «Заказы»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 25 — Цветное окно «Производство»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 26 — Цветная форма редактирования заказа

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 27 — Цветное окно «Администрирование»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 28 — Цветная форма редактирования профиля пользователя

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 29 — Цветное окно «Отчёты»

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 30 — Карта навигации приложения

# Задание 4. Разработка программы

## Реализация приложения

Программа для фабрики «Модный Пошив» была реализована на языке Python с подключением к БД PostgreSQL.

Структура приложения (рисунок 31).

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 31 — Структура проекта

Дальше идёт реализация окон приложения (рисунки 32–45).

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 32 — Окно авторизации

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 33 — Сообщение о неверном логине или пароле

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 34 — Сообщение об успешной авторизации

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 35 — Главная страница от лица администратора

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 36 — Главная страница от лица менеджера по продажам

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 37 — Главная страница от лица мастера цеха

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 38 — Главная страница от лица кладовщика

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 39 — Главная страница от лица клиента

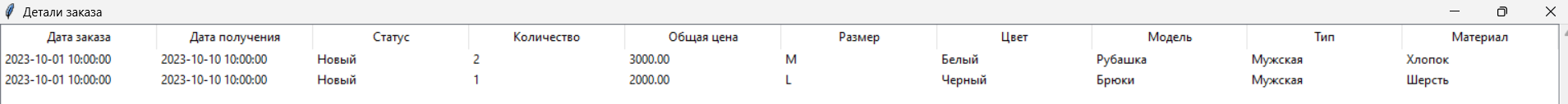


Рисунок 40 — Поиск товаров по номеру заказа

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 41 — Страница «Заказы»

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 42 — Форма добавления заказа

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 43 — Сообщение об ошибке, если не все поля заполнены в форме

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 44 — Форма редактирования заказа

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Автоматически созданное описание

Рисунок 45 — Возможность удалить заказ

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 46 — Подтверждение удаления заказа

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Автоматически созданное описание

Рисунок 47 — Страница «Производство» с таким же функционалом, как и страница «Заказы»

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 48 — Страница «Склад» с таким же функционалом, как и страница «Заказы»

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 49 — Страница «Отчёты»

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 50 — Сформировали «Отчёт по заказам»

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 51 — Сформировали «Отчёт по производству»

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 52 — Сформировали «Отчёт по материалам»

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 53 — Сформировали «Отчёт по готовой продукции»

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 54 — Страница «Администрирование» с таким же функционалом, как и страница «Заказы», где можно назначать роль пользователя

## Отладка приложения

В файле main\_window.py поставил точку остановки (рисунок 55).

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 55 — Точка остановки

Изображение выглядит как текст, Шрифт, программное обеспечение, снимок экрана

Автоматически созданное описание

Рисунок 55 — Видимые значения

# Задание 5. Тестирование

## Тестирование

Аннотация теста:

|  |  |
| --- | --- |
| Название проекта | FashionableTailoring |
| Рабочая версия | 1.0 |
| Имя тестирующего | Александр Михеев |
| Дата теста | 27.09.2024 |

Тестовый пример №1:

|  |  |
| --- | --- |
| Тестовый пример | TC\_UI\_1 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Проверка успешной авторизации |
| Краткое изложение теста | Проверка, что пользователь может успешно авторизоваться с действительным логином и паролем |
| Этапы теста | 1. Подготовка данных для теста. 2. Открытие окна авторизации 3. Заполнение полей «Логин» и «Пароль» тестовыми данными 4. Нажатие кнопки «Войти» 5. Ожидаемое сообщение об успешной авторизации 6. Нажимаем кнопку «ОК» 7. Проверка ожидаемых результатов. |
| Тестовые данные | * Логин: admin * Пароль: admin123 |
| Ожидаемый результат: | * Вызов метода execute\_query с правильным запросом. * Появление сообщения "Успех", "Авторизация прошла успешно!". * Вызов метода destroy. * Вызов метода open\_main\_window. |
| Фактический результат | Результат выполнения теста |
| Статус | Зачет |
| Предварительное условие | База данных должна содержать пользователя с логином "admin" и паролем "admin123" |
| Постусловие | Система должна перейти в главное окно после успешной авторизации |
| Примечания/комментарии |  |

Пошаговые действия (рисунки 56–58).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 56 — Заполняем поля «Логин» и «Пароль» тестовыми данными, нажимаем кнопку «Войти»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 57 — Ожидаемое сообщение об успешной авторизации, нажимаем кнопку «ОК»

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 58 — Видим ожидаемое окно «Главная»

Тестовый пример №2:

|  |  |
| --- | --- |
| Тестовый пример | TC\_UI\_2 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Проверка некорректности авторизации |
| Краткое изложение теста | Проверка, что пользователь не может успешно авторизоваться с недействительным логином и паролем |
| Этапы теста | 1. Подготовка данных для теста. 2. Открытие окна авторизации 3. Заполнение полей «Логин» и «Пароль» тестовыми данными 4. Нажатие кнопки «Войти» 5. Ожидаемое сообщение об ошибке 6. Проверка ожидаемых результатов. |
| Тестовые данные | * Логин: admin * Пароль: admin1234 |
| Ожидаемый результат: | * Вызов метода execute\_query с правильным запросом. * Появление сообщения "Ошибка", "Неверный логин или пароль". |
| Фактический результат | Результат выполнения теста |
| Статус | Зачет |
| Предварительное условие | База данных не должна содержать пользователя с логином "admin" и паролем "admin1234" |
| Постусловие | Система не должна перейти в главное окно после успешной авторизации |
| Примечания/комментарии |  |

Пошаговые действия (рисунки 59–60).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 59 — Заполняем поля «Логин» и «Пароль» тестовыми данными, нажимаем кнопку «Войти»

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 60 — Ожидаемое сообщение об ошибке

Тестовый пример №3:

|  |  |
| --- | --- |
| Тестовый пример | TC\_UI\_3 |
| Приоритет тестирования | Средний |
| Заголовок/название теста | Проверка открытия окна заказов |
| Краткое изложение теста | Проверка, что окно заказов открывается корректно. |
| Этапы теста | 1. Создание экземпляра MainWindow. 2. Вызов метода open\_orders. 3. Проверка, что OrdersWindow был вызван с правильными аргументами. |
| Тестовые данные | Информация о пользователе:   * Администратор * Иван Иванов * admin * admin123 * 1234567890 |
| Ожидаемый результат: | Вызов метода OrdersWindow с правильными аргументами. |
| Фактический результат | Результат выполнения теста |
| Статус | Зачет |
| Предварительное условие | Пользователь должен быть авторизован. |
| Постусловие | Окно заказов должно быть открыто. |
| Примечания/комментарии |  |

Пошаговые действия (рисунок 61).

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 61 — Окно «Заказы»

Тестовый пример №4:

|  |  |
| --- | --- |
| Тестовый пример | TC\_UI\_4 |
| Приоритет тестирования | Средний |
| Заголовок/название теста | Проверка отслеживания заказа |
| Краткое изложение теста | Проверка, что заказ может быть отслежен корректно |
| Этапы теста | 1. Подготовка данных для теста. 2. Создание экземпляра MainWindow. 3. Мокировние ввода данных. 4. Вызов метода track\_order. 5. Проверка, что execute\_query был вызван с правильными аргументами. 6. Проверка, что showerror не был вызван. 7. Проверка, что Toplevel был вызван. |
| Тестовые данные | * ID заказа: 1 |
| Ожидаемый результат: | * Вызов метода execute\_query с правильным запросом. * Отсутствие вызова метода showerror. * Вызов метода Toplevel. |
| Фактический результат | Результат выполнения теста |
| Статус | Зачет |
| Предварительное условие | * Пользователь должен быть авторизован. * База данных должна содержать заказ с ID. |
| Постусловие | Окно с информацией о заказе должно быть открыто. |
| Примечания/комментарии |  |

Пошаговые действия (рисунки 62–63).

Изображение выглядит как текст, снимок экрана, Шрифт, белый

Автоматически созданное описание

Рисунок 62 — Заполняем поле для отслеживания заказа



Рисунок 63 — Ожидаемый вывод дателей заказа

Тестовый пример №5:

|  |  |
| --- | --- |
| Тестовый пример | TC\_UI\_5 |
| Приоритет тестирования | Средний |
| Заголовок/название теста | Проверка отслеживания несуществующего заказа |
| Краткое изложение теста | Проверка, что попытка отслеживания несуществующего заказа приводит к ошибке. |
| Этапы теста | 1. Подготовка данных для теста. 2. Создание экземпляра MainWindow. 3. Мокирование ввода данных. 4. Вызов метода track\_order. 5. Проверка, что execute\_query был вызван с правильными аргументами. 6. Проверка, что showerror был вызван с правильными аргументами. |
| Тестовые данные | * ID заказа: 55 |
| Ожидаемый результат: | * Вызов метода execute\_query с правильным запросом. * Появление сообщения "Ошибка", "Заказ не найден". |
| Фактический результат | Результат выполнения теста |
| Статус | Зачет |
| Предварительное условие | * Пользователь должен быть авторизован. * База данных должна содержать заказ с ID. |
| Постусловие | Система должна вывести окно об ошибке |
| Примечания/комментарии |  |

Пошаговые действия (рисунки 64–65).

Изображение выглядит как текст, снимок экрана, Шрифт, белый

Автоматически созданное описание

Рисунок 62 — Заполняем поле для отслеживания заказа

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 63 — Вывод ошибки «Заказ не найден»

## Unit-тесты

1. test\_login\_success (Проверка успешной авторизации)
2. test\_login\_failure (Проверка неуспешной авторизации)
3. test\_open\_orders (Проверка открытия окна заказов)
4. test\_track\_order (Проверка отслеживания заказа)
5. test\_track\_order\_not\_found (Проверка отслеживания несуществующего заказа)
6. test\_load\_orders (Проверка загрузки заказов в OrdersWindow)
7. test\_generate\_orders\_report (Проверка генерации отчета по заказам в ReportsWindow)
8. test\_generate\_production\_report (Проверка генерации отчета по производству в ReportsWindow)
9. test\_generate\_materials\_report (Проверка генерации отчета по материалам в ReportsWindow)
10. test\_generate\_products\_report (Проверка генерации отчета по продуктам в ReportsWindow)

# Задание 6. GitHub. Оценка проекта

В конце своего проекта я выгрузил рабочее приложение на GitHub. Использование системы контроля версий Git позволяет отслеживать изменения в коде, возвращаться к предыдущим версиям, а также работать над проектом в команде. Выгрузка проекта в репозиторий Git позволяет легко делиться кодом с другими разработчиками, а также хранить проект в централизованном месте (рисунок 64).

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 64 — Страница GitHub

# ПРИЛОЖЕНИЕ А

## SQL-запросы

SQL-запросы для создания таблиц в PostgreSQL с каскадными взаимодействиями.

-- Создание таблицы Role

CREATE TABLE Role (

role\_id SERIAL PRIMARY KEY,

role\_name VARCHAR(255) UNIQUE

);

-- Создание таблицы User

CREATE TABLE User (

user\_id SERIAL PRIMARY KEY,

role\_id INT NOT NULL,

full\_name TEXT NOT NULL,

login TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

phone VARCHAR(20),

FOREIGN KEY (role\_id) REFERENCES Role(role\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Создание таблицы Material

CREATE TABLE Material (

material\_id SERIAL PRIMARY KEY,

material\_name VARCHAR(255) NOT NULL

);

-- Создание таблицы ProductType

CREATE TABLE ProductType (

type\_id SERIAL PRIMARY KEY,

type\_name VARCHAR(255) NOT NULL UNIQUE

);

-- Создание таблицы ProductModel

CREATE TABLE ProductModel (

model\_id SERIAL PRIMARY KEY,

model\_name VARCHAR(255) NOT NULL UNIQUE

);

-- Создание таблицы Product

CREATE TABLE Product (

product\_id SERIAL PRIMARY KEY,

type\_id INT NOT NULL,

model\_id INT NOT NULL,

size VARCHAR(10) NOT NULL,

color VARCHAR(20) NOT NULL,

material\_id INT NOT NULL,

unit\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (type\_id) REFERENCES ProductType(type\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (model\_id) REFERENCES ProductModel(model\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (material\_id) REFERENCES Material(material\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Создание таблицы Order

CREATE TABLE Order (

order\_id SERIAL PRIMARY KEY,

order\_date TIMESTAMP NOT NULL,

user\_id INT NOT NULL,

period\_of\_execution TIMESTAMP NOT NULL,

order\_status VARCHAR(20) NOT NULL,

FOREIGN KEY (user\_id) REFERENCES User(user\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Создание таблицы OrderLine

CREATE TABLE OrderLine (

order\_line\_id SERIAL PRIMARY KEY,

order\_id INT NOT NULL,

product\_id INT NOT NULL,

quantity INT DEFAULT 1,

total\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (order\_id) REFERENCES Order(order\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (product\_id) REFERENCES Product(product\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Создание таблицы ProductionTarget

CREATE TABLE ProductionTarget (

target\_id SERIAL PRIMARY KEY,

order\_id INT NOT NULL,

performer\_id INT NOT NULL,

period\_of\_execution TIMESTAMP NOT NULL,

target\_status VARCHAR(20) NOT NULL,

FOREIGN KEY (order\_id) REFERENCES Order(order\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (performer\_id) REFERENCES User(user\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Создание таблицы Warehouse

CREATE TABLE Warehouse (

product\_id INT PRIMARY KEY,

quantity INT DEFAULT 0,

FOREIGN KEY (product\_id) REFERENCES Product(product\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

SQL-запросы для заполнения таблиц данными в PostgreSQL.

INSERT INTO Role (role\_name) VALUES

('Администратор'),

('Менеджер по продажам'),

('Мастер цеха'),

('Кладовщик'),

(‘Клиент’),

INSERT INTO User (role\_id, full\_name, login, password, phone) VALUES

(1, 'Иван Иванов', 'admin', 'admin123', '1234567890'),

(2, 'Петр Петров', 'sales\_manager', 'sales123', '2345678901'),

(3, 'Анна Сидорова', 'workshop\_master', 'workshop123', '3456789012'),

(4, 'Елена Кузнецова', 'warehouse\_keeper', 'warehouse123', '4567890123'),

(5, 'Сергей Сергеев', 'director', 'director123', '5678901234'),

(6, 'Мария Иванова', 'accountant', 'accountant123', '6789012345'),

(7, 'Дмитрий Дмитриев', 'logistician', 'logistician123', '7890123456'),

(8, 'Ольга Ольгина', 'marketer', 'marketer123', '8901234567'),

(9, 'Алексей Алексеев', 'consultant', 'consultant123', '9012345678'),

(10, 'Екатерина Екатеринова', 'technologist', 'technologist123', '0123456789');

INSERT INTO User (role\_id, full\_name, login, password, phone) VALUES

(1, 'Иван Иванов', 'admin', 'admin123', '1234567890'),

(2, 'Петр Петров', 'sales\_manager', 'sales123', '2345678901'),

(3, 'Анна Сидорова', 'workshop\_master', 'workshop123', '3456789012'),

(4, 'Елена Кузнецова', 'warehouse\_keeper', 'warehouse123', '4567890123'),

(5, 'Сергей Сергеев', 'director', 'director123', '5678901234'),

(6, 'Мария Иванова', 'accountant', 'accountant123', '6789012345'),

(7, 'Дмитрий Дмитриев', 'logistician', 'logistician123', '7890123456'),

(8, 'Ольга Ольгина', 'marketer', 'marketer123', '8901234567'),

(9, 'Алексей Алексеев', 'consultant', 'consultant123', '9012345678'),

(10, 'Екатерина Екатеринова', 'technologist', 'technologist123', '0123456789');

INSERT INTO Material (material\_name) VALUES

('Хлопок'),

('Шерсть'),

('Полиэстер'),

('Лен'),

('Шелк'),

('Вискоза'),

('Кашемир'),

('Шифон'),

('Деним'),

('Кожа');

INSERT INTO ProductType (type\_name) VALUES

('Мужская'),

('Женская'),

('Детская'),

('Подростковая'),

('Безразмерная'),

('Спецодежда'),

('Спортивная'),

('Вечерняя'),

('Офисная'),

('Пляжная');

INSERT INTO ProductModel (model\_name) VALUES

('Рубашка'),

('Брюки'),

('Платье'),

('Футболка'),

('Юбка'),

('Пиджак'),

('Блузка'),

('Шорты'),

('Комбинезон'),

('Сарафан');

INSERT INTO Product (type\_id, model\_id, size, color, material\_id, unit\_price) VALUES

(1, 1, 'M', 'Белый', 1, 1500.00),

(1, 2, 'L', 'Черный', 2, 2000.00),

(2, 3, 'S', 'Красный', 3, 2500.00),

(2, 4, 'M', 'Синий', 4, 1000.00),

(3, 1, 'S', 'Зеленый', 1, 800.00),

(3, 2, 'M', 'Желтый', 2, 900.00),

(1, 3, 'L', 'Серый', 3, 1800.00),

(2, 1, 'S', 'Розовый', 4, 1200.00),

(3, 4, 'M', 'Оранжевый', 1, 700.00),

(1, 4, 'L', 'Фиолетовый', 2, 1300.00);

INSERT INTO Order (order\_date, user\_id, period\_of\_execution, order\_status) VALUES

('2023-10-01 10:00:00', 2, '2023-10-10 10:00:00', 'Новый'),

('2023-10-02 11:00:00', 2, '2023-10-11 11:00:00', 'В работе'),

('2023-10-03 12:00:00', 2, '2023-10-12 12:00:00', 'Готов'),

('2023-10-04 13:00:00', 2, '2023-10-13 13:00:00', 'Отгружен'),

('2023-10-05 14:00:00', 2, '2023-10-14 14:00:00', 'Новый'),

('2023-10-06 15:00:00', 2, '2023-10-15 15:00:00', 'В работе'),

('2023-10-07 16:00:00', 2, '2023-10-16 16:00:00', 'Готов'),

('2023-10-08 17:00:00', 2, '2023-10-17 17:00:00', 'Отгружен'),

('2023-10-09 18:00:00', 2, '2023-10-18 18:00:00', 'Новый'),

('2023-10-10 19:00:00', 2, '2023-10-19 19:00:00', 'В работе');

INSERT INTO OrderLine (order\_id, product\_id, quantity, total\_price) VALUES

(1, 1, 2, 3000.00),

(1, 2, 1, 2000.00),

(2, 3, 3, 7500.00),

(2, 4, 2, 2000.00),

(3, 5, 1, 800.00),

(3, 6, 2, 1800.00),

(4, 7, 3, 5400.00),

(4, 8, 1, 1200.00),

(5, 9, 2, 1400.00),

(5, 10, 1, 1300.00);

INSERT INTO ProductionTarget (order\_id, performer\_id, period\_of\_execution, target\_status) VALUES

(1, 3, '2023-10-10 10:00:00', 'В работе'),

(2, 3, '2023-10-11 11:00:00', 'В работе'),

(3, 3, '2023-10-12 12:00:00', 'Выполнено'),

(4, 3, '2023-10-13 13:00:00', 'Выполнено'),

(5, 3, '2023-10-14 14:00:00', 'В работе'),

(6, 3, '2023-10-15 15:00:00', 'В работе'),

(7, 3, '2023-10-16 16:00:00', 'Выполнено'),

(8, 3, '2023-10-17 17:00:00', 'Выполнено'),

(9, 3, '2023-10-18 18:00:00', 'В работе'),

(10, 3, '2023-10-19 19:00:00', 'В работе');

INSERT INTO Warehouse (product\_id, quantity) VALUES

(1, 10),

(2, 15),

(3, 20),

(4, 25),

(5, 30),

(6, 35),

(7, 40),

(8, 45),

(9, 50),

(10, 55);

## Код приложения

import tkinter as tk  
from ui.login\_window import LoginWindow  
  
def main():  
 root = tk.Tk()  
 app = LoginWindow(root)  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

import psycopg2  
  
class Database:  
 def \_\_init\_\_(self):  
 self.conn = psycopg2.connect(  
 database="FashionableTailoring",  
 user="postgres",  
 password="1525",  
 host="127.0.0.1",  
 port="5432"  
 )  
  
 def execute\_query(self, query, params=None, fetch=True):  
 self.cursor = self.conn.cursor()  
 if params:  
 self.cursor.execute(query, params)  
 else:  
 self.cursor.execute(query)  
  
 if fetch:  
 result = self.cursor.fetchall()  
 else:  
 result = None  
  
 self.conn.commit()  
 self.cursor.close()  
 return result  
  
 def close(self):  
 self.cursor.close()  
 self.conn.close()

import tkinter as tk  
from tkinter import ttk, messagebox  
from database import Database  
  
class AdministrationWindow:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.db = Database()  
 self.window = tk.Toplevel(self.root)  
 self.window.title("Администрирование")  
 self.window.geometry("800x600")  
  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 tk.Label(self.window, text="Администрирование", font=("Arial", 18)).pack(pady=20)  
  
 # Создаем таблицу для отображения данных пользователей  
 self.tree = ttk.Treeview(self.window, columns=(  
 "user\_id", "role\_name", "full\_name", "login", "password", "phone"), show="headings")  
  
 # Определяем заголовки столбцов  
 self.tree.heading("user\_id", text="ID пользователя")  
 self.tree.heading("role\_name", text="Роль")  
 self.tree.heading("full\_name", text="ФИО")  
 self.tree.heading("login", text="Логин")  
 self.tree.heading("password", text="Пароль")  
 self.tree.heading("phone", text="Телефон")  
  
 # Устанавливаем ширину столбцов  
 self.tree.column("user\_id", width=100)  
 self.tree.column("role\_name", width=100)  
 self.tree.column("full\_name", width=150)  
 self.tree.column("login", width=100)  
 self.tree.column("password", width=100)  
 self.tree.column("phone", width=150)  
  
 self.tree.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем полосу прокрутки  
 scrollbar = ttk.Scrollbar(self.window, orient="vertical", command=self.tree.yview)  
 self.tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.load\_users()  
  
 tk.Button(self.window, text="Добавить пользователя", command=self.add\_user).pack(pady=10)  
 tk.Button(self.window, text="Редактировать пользователя", command=self.edit\_user).pack(pady=10)  
 tk.Button(self.window, text="Удалить пользователя", command=self.delete\_user).pack(pady=10)  
  
 def load\_users(self):  
 # Очищаем таблицу  
 for i in self.tree.get\_children():  
 self.tree.delete(i)  
  
 # Загружаем данные пользователей из базы данных  
 users = self.db.execute\_query("SELECT \  
 Users.user\_id, \  
 Roles.role\_name, \  
 Users.full\_name, \  
 Users.login, \  
 Users.password, \  
 Users.phone \  
 FROM Users \  
 JOIN Roles ON Users.role\_id = Roles.role\_id")  
 for user in users:  
 self.tree.insert("", "end", values=user)  
  
 def add\_user(self):  
 # Создаем диалоговое окно для добавления пользователя  
 add\_window = tk.Toplevel(self.window)  
 add\_window.title("Добавить пользователя")  
 add\_window.geometry("400x300")  
  
 tk.Label(add\_window, text="Роль:").pack(pady=5)  
 role\_name\_entry = tk.Entry(add\_window)  
 role\_name\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="ФИО:").pack(pady=5)  
 full\_name\_entry = tk.Entry(add\_window)  
 full\_name\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Логин:").pack(pady=5)  
 login\_entry = tk.Entry(add\_window)  
 login\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Пароль:").pack(pady=5)  
 password\_entry = tk.Entry(add\_window)  
 password\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Телефон:").pack(pady=5)  
 phone\_entry = tk.Entry(add\_window)  
 phone\_entry.pack(pady=5)  
  
 def save\_user():  
 role\_name = role\_name\_entry.get()  
 full\_name = full\_name\_entry.get()  
 login = login\_entry.get()  
 password = password\_entry.get()  
 phone = phone\_entry.get()  
  
 if not role\_name or not full\_name or not login or not password or not phone:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 # Получаем role\_id по имени роли  
 role\_id = self.db.execute\_query("SELECT role\_id FROM Roles WHERE role\_name = %s", (role\_name,))  
 if not role\_id:  
 messagebox.showerror("Ошибка", "Роль не найдена")  
 return  
  
 self.db.execute\_query(  
 "INSERT INTO Users (role\_id, full\_name, login, password, phone) VALUES (%s, %s, %s, %s, %s)",  
 (role\_id[0][0], full\_name, login, password, phone), fetch=False)  
 add\_window.destroy()  
 self.load\_users()  
  
 tk.Button(add\_window, text="Сохранить", command=save\_user).pack(pady=10)  
  
 def edit\_user(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите пользователя для редактирования")  
 return  
  
 user\_id = self.tree.item(selected\_item)['values'][0]  
 role\_name = self.tree.item(selected\_item)['values'][1]  
 full\_name = self.tree.item(selected\_item)['values'][2]  
 login = self.tree.item(selected\_item)['values'][3]  
 password = self.tree.item(selected\_item)['values'][4]  
 phone = self.tree.item(selected\_item)['values'][5]  
  
 # Создаем диалоговое окно для редактирования пользователя  
 edit\_window = tk.Toplevel(self.window)  
 edit\_window.title("Редактировать пользователя")  
 edit\_window.geometry("400x300")  
  
 tk.Label(edit\_window, text="Роль:").pack(pady=5)  
 role\_name\_entry = tk.Entry(edit\_window)  
 role\_name\_entry.insert(0, role\_name)  
 role\_name\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="ФИО:").pack(pady=5)  
 full\_name\_entry = tk.Entry(edit\_window)  
 full\_name\_entry.insert(0, full\_name)  
 full\_name\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Логин:").pack(pady=5)  
 login\_entry = tk.Entry(edit\_window)  
 login\_entry.insert(0, login)  
 login\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Пароль:").pack(pady=5)  
 password\_entry = tk.Entry(edit\_window)  
 password\_entry.insert(0, password)  
 password\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Телефон:").pack(pady=5)  
 phone\_entry = tk.Entry(edit\_window)  
 phone\_entry.insert(0, phone)  
 phone\_entry.pack(pady=5)  
  
 def update\_user():  
 new\_role\_name = role\_name\_entry.get()  
 new\_full\_name = full\_name\_entry.get()  
 new\_login = login\_entry.get()  
 new\_password = password\_entry.get()  
 new\_phone = phone\_entry.get()  
  
 if not new\_role\_name or not new\_full\_name or not new\_login or not new\_password or not new\_phone:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 # Получаем role\_id по имени роли  
 role\_id = self.db.execute\_query("SELECT role\_id FROM Roles WHERE role\_name = %s", (new\_role\_name,))  
 if not role\_id:  
 messagebox.showerror("Ошибка", "Роль не найдена")  
 return  
  
 self.db.execute\_query(  
 "UPDATE Users SET role\_id = %s, full\_name = %s, login = %s, password = %s, phone = %s WHERE user\_id = %s",  
 (role\_id[0][0], new\_full\_name, new\_login, new\_password, new\_phone, user\_id), fetch=False)  
 edit\_window.destroy()  
 self.load\_users()  
  
 tk.Button(edit\_window, text="Сохранить", command=update\_user).pack(pady=10)  
  
 def delete\_user(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите пользователя для удаления")  
 return  
  
 user\_id = self.tree.item(selected\_item)['values'][0]  
 confirm = messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить пользователя №{user\_id}?")  
 if confirm:  
 self.db.execute\_query("DELETE FROM Users WHERE user\_id = %s", (user\_id,), fetch=False)  
 self.load\_users()

import tkinter as tk  
from tkinter import messagebox  
from ui.main\_window import MainWindow  
from database import Database  
  
  
class LoginWindow:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.db = Database()  
 self.root.title("Авторизация")  
 self.root.geometry("300x150")  
  
 # Создание виджетов  
 self.label\_login = tk.Label(root, text="Логин:")  
 self.label\_login.pack()  
  
 self.entry\_login = tk.Entry(root)  
 self.entry\_login.pack()  
  
 self.label\_password = tk.Label(root, text="Пароль:")  
 self.label\_password.pack()  
  
 self.entry\_password = tk.Entry(root, show="\*")  
 self.entry\_password.pack()  
  
 self.button\_login = tk.Button(root, text="Войти", command=self.login)  
 self.button\_login.pack()  
  
 def login(self):  
 # Получение данных из полей ввода  
 login = self.entry\_login.get()  
 password = self.entry\_password.get()  
  
 # Подключение к базе данных  
 user = self.db.execute\_query(f"SELECT Users.user\_id, Roles.role\_name, Users.full\_name, Users.login, Users.password, Users.phone FROM Users \  
 JOIN Roles ON Users.role\_id = Roles.role\_id \  
 WHERE login = '{login}' and password = '{password}'")  
  
 if user:  
 messagebox.showinfo("Успех", "Авторизация прошла успешно!")  
 self.root.destroy() # Закрываем окно авторизации  
 self.open\_main\_window(user) # Открываем главное окно  
 else:  
 messagebox.showerror("Ошибка", "Неверный логин или пароль")  
  
 def open\_main\_window(self, user\_info):  
 root = tk.Tk()  
 from ui.main\_window import MainWindow # Импортируем здесь, чтобы избежать циклического импорта  
 app = MainWindow(root, user\_info)  
 root.mainloop()

import tkinter as tk  
from tkinter import messagebox, ttk  
import psycopg2  
from ui.orders\_window import OrdersWindow  
from ui.production\_window import ProductionWindow  
from ui.warehouse\_window import WarehouseWindow  
from ui.reports\_window import ReportsWindow  
from ui.administration\_window import AdministrationWindow  
from database import Database  
  
class MainWindow:  
 def \_\_init\_\_(self, root, user\_info):  
 self.root = root  
 self.db = Database()  
 self.root.title("Главная")  
 self.root.geometry("800x600")  
 self.user\_info = user\_info[0]  
  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 # Создаем фрейм для шапки  
 header\_frame = tk.Frame(self.root)  
 header\_frame.pack(side=tk.TOP, fill=tk.X)  
  
 # Заголовок слева от шапки  
 tk.Label(header\_frame, text="Модный пошив", font=("Arial", 24)).pack(side=tk.LEFT, padx=10, pady=10)  
  
 # Получаем роль пользователя  
 role\_name = self.user\_info[1]  
  
 # Создаем кнопки в шапке в зависимости от роли  
 if role\_name == "Администратор":  
 tk.Button(header\_frame, text="Заказы", command=self.open\_orders).pack(side=tk.LEFT, padx=10, pady=10)  
 tk.Button(header\_frame, text="Производство", command=self.open\_production).pack(side=tk.LEFT, padx=10, pady=10)  
 tk.Button(header\_frame, text="Склад", command=self.open\_warehouse).pack(side=tk.LEFT, padx=10, pady=10)  
 tk.Button(header\_frame, text="Отчеты", command=self.open\_reports).pack(side=tk.LEFT, padx=10, pady=10)  
 tk.Button(header\_frame, text="Администрирование", command=self.open\_administration).pack(side=tk.LEFT, padx=10, pady=10)  
 elif role\_name == "Менеджер по продажам":  
 tk.Button(header\_frame, text="Заказы", command=self.open\_orders).pack(side=tk.LEFT, padx=10, pady=10)  
 elif role\_name == "Мастер цеха":  
 tk.Button(header\_frame, text="Производство", command=self.open\_production).pack(side=tk.LEFT, padx=10, pady=10)  
 elif role\_name == "Кладовщик":  
 tk.Button(header\_frame, text="Склад", command=self.open\_warehouse).pack(side=tk.LEFT, padx=10, pady=10)  
  
 # Создаем фрейм для основного содержимого  
 main\_frame = tk.Frame(self.root)  
 main\_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)  
  
 # Картинка человечка (профиль аккаунта)  
 profile\_image = tk.PhotoImage(file="profile.png") # Замените на путь к вашей картинке  
 profile\_image = profile\_image.subsample(5, 5) # Уменьшаем изображение в 5 раз  
 profile\_label = tk.Label(main\_frame, image=profile\_image)  
 profile\_label.image = profile\_image # Сохраняем ссылку на изображение, чтобы оно не удалялось сборщиком мусора  
 profile\_label.pack(pady=10)  
  
 # Имя и фамилия  
 tk.Label(main\_frame, text=f"Имя: {self.user\_info[2]}", font=("Arial", 12)).pack(pady=5)  
  
 # Роль  
 tk.Label(main\_frame, text=f"Роль: {role\_name}", font=("Arial", 12)).pack(pady=5)  
  
 # Логин через @  
 tk.Label(main\_frame, text=f"Логин: @{self.user\_info[3]}", font=("Arial", 12)).pack(pady=5)  
  
 # Поле "Отследить заказ"  
 tk.Label(main\_frame, text="Отследить заказ:", font=("Arial", 16)).pack(pady=10)  
 self.order\_entry = tk.Entry(main\_frame, font=("Arial", 12))  
 self.order\_entry.pack(pady=10)  
 tk.Button(main\_frame, text="Отследить", command=self.track\_order).pack(pady=10)  
  
 def open\_orders(self):  
 OrdersWindow(self.root, self.user\_info)  
  
 def open\_production(self):  
 ProductionWindow(self.root, self.user\_info)  
  
 def open\_warehouse(self):  
 WarehouseWindow(self.root)  
  
 def open\_reports(self):  
 ReportsWindow(self.root)  
  
 def open\_administration(self):  
 AdministrationWindow(self.root)  
  
 def track\_order(self):  
 order\_id = self.order\_entry.get()  
 if not order\_id:  
 messagebox.showerror("Ошибка", "Введите номер заказа")  
 return  
  
 try:  
 # Запрос к базе данных  
 query = """  
 SELECT Orders.order\_date, Orders.period\_of\_execution, Orders.order\_status,   
 OrderLines.quantity, OrderLines.total\_price, Products.size, Products.color,   
 ProductModels.model\_name, ProductTypes.type\_name, Materials.material\_name  
 FROM Orders  
 JOIN OrderLines ON Orders.order\_id = OrderLines.order\_id  
 JOIN Products ON OrderLines.product\_id = Products.product\_id  
 JOIN ProductModels ON Products.model\_id = ProductModels.model\_id  
 JOIN ProductTypes ON Products.type\_id = ProductTypes.type\_id  
 JOIN Materials ON Products.material\_id = Materials.material\_id  
 WHERE Orders.order\_id = %s;  
 """  
 orders = self.db.execute\_query(query, (order\_id,))  
  
 if orders:  
 self.display\_order\_details(orders)  
 else:  
 messagebox.showerror("Ошибка", "Заказ не найден")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Ошибка подключения к базе данных: {e}")  
  
 def display\_order\_details(self, orders):  
 # Создаем новое окно для отображения деталей заказа  
 order\_window = tk.Toplevel(self.root)  
 order\_window.title("Детали заказа")  
 order\_window.geometry("800x600")  
  
 # Создаем таблицу для отображения данных  
 tree = ttk.Treeview(order\_window, columns=(  
 "order\_date", "period\_of\_execution", "order\_status", "quantity", "total\_price", "size", "color",  
 "model\_name", "type\_name", "material\_name"), show="headings")  
  
 # Определяем заголовки столбцов  
 tree.heading("order\_date", text="Дата заказа")  
 tree.heading("period\_of\_execution", text="Дата получения")  
 tree.heading("order\_status", text="Статус")  
 tree.heading("quantity", text="Количество")  
 tree.heading("total\_price", text="Общая цена")  
 tree.heading("size", text="Размер")  
 tree.heading("color", text="Цвет")  
 tree.heading("model\_name", text="Модель")  
 tree.heading("type\_name", text="Тип")  
 tree.heading("material\_name", text="Материал")  
  
 # Устанавливаем ширину столбцов  
 tree.column("order\_date", width=100)  
 tree.column("period\_of\_execution", width=100)  
 tree.column("order\_status", width=100)  
 tree.column("quantity", width=100)  
 tree.column("total\_price", width=100)  
 tree.column("size", width=100)  
 tree.column("color", width=100)  
 tree.column("model\_name", width=100)  
 tree.column("type\_name", width=100)  
 tree.column("material\_name", width=100)  
  
 # Вставляем данные в таблицу  
 for order in orders:  
 tree.insert("", "end", values=order)  
  
 # Добавляем полосу прокрутки  
 scrollbar = ttk.Scrollbar(order\_window, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 tree.pack(fill=tk.BOTH, expand=True)

import tkinter as tk  
from tkinter import ttk, messagebox  
from database import Database  
from datetime import datetime  
  
class OrdersWindow:  
 def \_\_init\_\_(self, root, user\_info):  
 self.root = root  
 self.db = Database()  
 self.window = tk.Toplevel(self.root)  
 self.window.title("Заказы")  
 self.window.geometry("800x600")  
 self.user\_info = user\_info  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 tk.Label(self.window, text="Заказы", font=("Arial", 18)).pack(pady=20)  
  
 # Создаем таблицу для отображения заказов  
 self.tree = ttk.Treeview(self.window, columns=(  
 "order\_id", "full\_name", "order\_date", "period\_of\_execution", "order\_status"), show="headings")  
  
 # Определяем заголовки столбцов  
 self.tree.heading("order\_id", text="Номер заказа")  
 self.tree.heading("full\_name", text="Пользователь")  
 self.tree.heading("order\_date", text="Дата заказа")  
 self.tree.heading("period\_of\_execution", text="Дата получения")  
 self.tree.heading("order\_status", text="Статус")  
  
 # Устанавливаем ширину столбцов  
 self.tree.column("order\_id", width=100)  
 self.tree.column("full\_name", width=150)  
 self.tree.column("order\_date", width=150)  
 self.tree.column("period\_of\_execution", width=150)  
 self.tree.column("order\_status", width=150)  
  
 self.tree.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем полосу прокрутки  
 scrollbar = ttk.Scrollbar(self.window, orient="vertical", command=self.tree.yview)  
 self.tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.load\_orders()  
  
 tk.Button(self.window, text="Добавить заказ", command=self.add\_order).pack(pady=10)  
 tk.Button(self.window, text="Редактировать заказ", command=self.edit\_order).pack(pady=10)  
 tk.Button(self.window, text="Удалить заказ", command=self.delete\_order).pack(pady=10)  
  
 def load\_orders(self):  
 # Очищаем таблицу  
 for i in self.tree.get\_children():  
 self.tree.delete(i)  
  
 # Загружаем заказы из базы данных  
 orders = self.db.execute\_query("SELECT \  
 Orders.order\_id, \  
 Users.full\_name, \  
 Orders.order\_date, \  
 Orders.period\_of\_execution, \  
 Orders.order\_status \  
 FROM Orders \  
 JOIN Users ON Users.user\_id = Orders.user\_id")  
 for order in orders:  
 self.tree.insert("", "end", values=order)  
  
 def add\_order(self):  
 # Создаем диалоговое окно для добавления заказа  
 add\_window = tk.Toplevel(self.window)  
 add\_window.title("Добавить заказ")  
 add\_window.geometry("400x300")  
  
 tk.Label(add\_window, text="Дата заказа:").pack(pady=5)  
 order\_date\_entry = tk.Entry(add\_window)  
 order\_date\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Дата получения:").pack(pady=5)  
 period\_of\_execution\_entry = tk.Entry(add\_window)  
 period\_of\_execution\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Статус:").pack(pady=5)  
 order\_status\_entry = tk.Entry(add\_window)  
 order\_status\_entry.pack(pady=5)  
  
 def save\_order():  
 order\_date = order\_date\_entry.get()  
 period\_of\_execution = period\_of\_execution\_entry.get()  
 order\_status = order\_status\_entry.get()  
  
 if not order\_date or not period\_of\_execution or not order\_status:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 # Убедитесь, что даты передаются в правильном формате  
 try:  
 order\_date = datetime.strptime(order\_date, "%Y-%m-%d %H:%M:%S").strftime("%Y-%m-%d %H:%M:%S")  
 period\_of\_execution = datetime.strptime(period\_of\_execution, "%Y-%m-%d %H:%M:%S").strftime(  
 "%Y-%m-%d %H:%M:%S")  
 except ValueError:  
 messagebox.showerror("Ошибка", "Неверный формат даты. Используйте формат YYYY-MM-DD HH:MM:SS")  
 return  
  
 self.db.execute\_query(  
 "INSERT INTO Orders (order\_date, user\_id, period\_of\_execution, order\_status) VALUES (%s, %s, %s, %s)",  
 (order\_date, self.user\_info[0], period\_of\_execution, order\_status), fetch=False)  
 add\_window.destroy()  
 self.load\_orders()  
  
 tk.Button(add\_window, text="Сохранить", command=save\_order).pack(pady=10)  
  
 def edit\_order(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите заказ для редактирования")  
 return  
  
 order\_id = self.tree.item(selected\_item)['values'][0]  
 order\_date = self.tree.item(selected\_item)['values'][2]  
 period\_of\_execution = self.tree.item(selected\_item)['values'][3]  
 order\_status = self.tree.item(selected\_item)['values'][4]  
  
 # Создаем диалоговое окно для редактирования заказа  
 edit\_window = tk.Toplevel(self.window)  
 edit\_window.title("Редактировать заказ")  
 edit\_window.geometry("400x300")  
  
 tk.Label(edit\_window, text="Дата заказа:").pack(pady=5)  
 order\_date\_entry = tk.Entry(edit\_window)  
 order\_date\_entry.insert(0, order\_date)  
 order\_date\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Дата получения:").pack(pady=5)  
 period\_of\_execution\_entry = tk.Entry(edit\_window)  
 period\_of\_execution\_entry.insert(0, period\_of\_execution)  
 period\_of\_execution\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Статус:").pack(pady=5)  
 order\_status\_entry = tk.Entry(edit\_window)  
 order\_status\_entry.insert(0, order\_status)  
 order\_status\_entry.pack(pady=5)  
  
 def update\_order():  
 new\_order\_date = order\_date\_entry.get()  
 new\_period\_of\_execution = period\_of\_execution\_entry.get()  
 new\_order\_status = order\_status\_entry.get()  
  
 if not new\_order\_date or not new\_period\_of\_execution or not new\_order\_status:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 # Убедитесь, что даты передаются в правильном формате  
 try:  
 new\_order\_date = datetime.strptime(new\_order\_date, "%Y-%m-%d %H:%M:%S").strftime("%Y-%m-%d %H:%M:%S")  
 new\_period\_of\_execution = datetime.strptime(new\_period\_of\_execution, "%Y-%m-%d %H:%M:%S").strftime(  
 "%Y-%m-%d %H:%M:%S")  
 except ValueError:  
 messagebox.showerror("Ошибка", "Неверный формат даты. Используйте формат YYYY-MM-DD HH:MM:SS")  
 return  
  
 self.db.execute\_query(  
 "UPDATE Orders SET order\_date = %s, period\_of\_execution = %s, order\_status = %s WHERE order\_id = %s",  
 (new\_order\_date, new\_period\_of\_execution, new\_order\_status, order\_id), fetch=False)  
 edit\_window.destroy()  
 self.load\_orders()  
  
 tk.Button(edit\_window, text="Сохранить", command=update\_order).pack(pady=10)  
  
 def delete\_order(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите заказ для удаления")  
 return  
  
 order\_id = self.tree.item(selected\_item)['values'][0]  
 confirm = messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить заказ №{order\_id}?")  
 if confirm:  
 self.db.execute\_query("DELETE FROM Orders WHERE order\_id = %s", (order\_id,), fetch=False)  
 self.load\_orders()

import tkinter as tk  
from tkinter import ttk, messagebox  
from database import Database  
from datetime import datetime  
  
class ProductionWindow:  
 def \_\_init\_\_(self, root, user\_info):  
 self.root = root  
 self.db = Database()  
 self.window = tk.Toplevel(self.root)  
 self.window.title("Производство")  
 self.window.geometry("800x600")  
 self.user\_info = user\_info  
  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 tk.Label(self.window, text="Производство", font=("Arial", 18)).pack(pady=20)  
  
 # Создаем таблицу для отображения заданий  
 self.tree = ttk.Treeview(self.window, columns=(  
 "target\_id", "order\_id", "performer\_name", "period\_of\_execution", "target\_status"), show="headings")  
  
 # Определяем заголовки столбцов  
 self.tree.heading("target\_id", text="Номер задания")  
 self.tree.heading("order\_id", text="Номер заказа")  
 self.tree.heading("performer\_name", text="Исполнитель")  
 self.tree.heading("period\_of\_execution", text="Дата получения")  
 self.tree.heading("target\_status", text="Статус")  
  
 # Устанавливаем ширину столбцов  
 self.tree.column("target\_id", width=100)  
 self.tree.column("order\_id", width=100)  
 self.tree.column("performer\_name", width=150)  
 self.tree.column("period\_of\_execution", width=300)  
 self.tree.column("target\_status", width=150)  
  
 self.tree.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем полосу прокрутки  
 scrollbar = ttk.Scrollbar(self.window, orient="vertical", command=self.tree.yview)  
 self.tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.load\_targets()  
  
 tk.Button(self.window, text="Добавить задание", command=self.add\_target).pack(pady=10)  
 tk.Button(self.window, text="Редактировать задание", command=self.edit\_target).pack(pady=10)  
 tk.Button(self.window, text="Удалить задание", command=self.delete\_target).pack(pady=10)  
  
 def load\_targets(self):  
 # Очищаем таблицу  
 for i in self.tree.get\_children():  
 self.tree.delete(i)  
  
 # Загружаем задания из базы данных  
 targets = self.db.execute\_query("SELECT \  
 ProductionTargets.target\_id, \  
 ProductionTargets.order\_id, \  
 Users.full\_name AS performer\_name, \  
 ProductionTargets.period\_of\_execution, \  
 ProductionTargets.target\_status \  
 FROM ProductionTargets \  
 JOIN Users ON Users.user\_id = ProductionTargets.performer\_id")  
 for target in targets:  
 self.tree.insert("", "end", values=target)  
  
 def add\_target(self):  
 # Создаем диалоговое окно для добавления задания  
 add\_window = tk.Toplevel(self.window)  
 add\_window.title("Добавить задание")  
 add\_window.geometry("400x300")  
  
 tk.Label(add\_window, text="Номер заказа:").pack(pady=5)  
 order\_id\_entry = tk.Entry(add\_window)  
 order\_id\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Дата получения:").pack(pady=5)  
 period\_of\_execution\_entry = tk.Entry(add\_window)  
 period\_of\_execution\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Статус:").pack(pady=5)  
 target\_status\_entry = tk.Entry(add\_window)  
 target\_status\_entry.pack(pady=5)  
  
 def save\_target():  
 order\_id = order\_id\_entry.get()  
 performer\_id = self.user\_info[0]  
 period\_of\_execution = period\_of\_execution\_entry.get()  
 target\_status = target\_status\_entry.get()  
  
 if not order\_id or not performer\_id or not period\_of\_execution or not target\_status:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 try:  
 period\_of\_execution = datetime.strptime(period\_of\_execution, "%Y-%m-%d %H:%M:%S").strftime(  
 "%Y-%m-%d %H:%M:%S")  
 except ValueError:  
 messagebox.showerror("Ошибка", "Неверный формат даты. Используйте формат YYYY-MM-DD HH:MM:SS")  
 return  
  
 self.db.execute\_query(  
 "INSERT INTO ProductionTargets (order\_id, performer\_id, period\_of\_execution, target\_status) VALUES (%s, %s, %s, %s)",  
 (order\_id, performer\_id, period\_of\_execution, target\_status), fetch=False)  
 add\_window.destroy()  
 self.load\_targets()  
  
 tk.Button(add\_window, text="Сохранить", command=save\_target).pack(pady=10)  
  
 def edit\_target(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите задание для редактирования")  
 return  
  
 target\_id = self.tree.item(selected\_item)['values'][0]  
 order\_id = self.tree.item(selected\_item)['values'][1]  
 performer\_id = self.user\_info[0]  
 period\_of\_execution = self.tree.item(selected\_item)['values'][3]  
 target\_status = self.tree.item(selected\_item)['values'][4]  
  
 # Создаем диалоговое окно для редактирования задания  
 edit\_window = tk.Toplevel(self.window)  
 edit\_window.title("Редактировать задание")  
 edit\_window.geometry("400x300")  
  
 tk.Label(edit\_window, text="Номер заказа:").pack(pady=5)  
 order\_id\_entry = tk.Entry(edit\_window)  
 order\_id\_entry.insert(0, order\_id)  
 order\_id\_entry.pack(pady=5)  
  
  
 tk.Label(edit\_window, text="Дата получения:").pack(pady=5)  
 period\_of\_execution\_entry = tk.Entry(edit\_window)  
 period\_of\_execution\_entry.insert(0, period\_of\_execution)  
 period\_of\_execution\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Статус:").pack(pady=5)  
 target\_status\_entry = tk.Entry(edit\_window)  
 target\_status\_entry.insert(0, target\_status)  
 target\_status\_entry.pack(pady=5)  
  
 def update\_target():  
 new\_order\_id = order\_id\_entry.get()  
 new\_period\_of\_execution = period\_of\_execution\_entry.get()  
 new\_target\_status = target\_status\_entry.get()  
  
 if not new\_order\_id or not performer\_id or not new\_period\_of\_execution or not new\_target\_status:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 try:  
 new\_period\_of\_execution = datetime.strptime(new\_period\_of\_execution, "%Y-%m-%d %H:%M:%S").strftime(  
 "%Y-%m-%d %H:%M:%S")  
 except ValueError:  
 messagebox.showerror("Ошибка", "Неверный формат даты. Используйте формат YYYY-MM-DD HH:MM:SS")  
 return  
  
 self.db.execute\_query(  
 "UPDATE ProductionTargets SET order\_id = %s, performer\_id = %s, period\_of\_execution = %s, target\_status = %s WHERE target\_id = %s",  
 (new\_order\_id, performer\_id, new\_period\_of\_execution, new\_target\_status, target\_id), fetch=False)  
 edit\_window.destroy()  
 self.load\_targets()  
  
 tk.Button(edit\_window, text="Сохранить", command=update\_target).pack(pady=10)  
  
 def delete\_target(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите задание для удаления")  
 return  
  
 target\_id = self.tree.item(selected\_item)['values'][0]  
 confirm = messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить задание №{target\_id}?")  
 if confirm:  
 self.db.execute\_query("DELETE FROM ProductionTargets WHERE target\_id = %s", (target\_id,), fetch=False)  
 self.load\_targets()

import tkinter as tk  
from tkinter import ttk, messagebox  
from database import Database  
  
class ReportsWindow:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.db = Database()  
 self.window = tk.Toplevel(self.root)  
 self.window.title("Отчеты")  
 self.window.geometry("800x600")  
  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 tk.Label(self.window, text="Отчеты", font=("Arial", 18)).pack(pady=20)  
  
 self.report\_type = tk.StringVar(value="orders")  
 tk.Radiobutton(self.window, text="Заказы", variable=self.report\_type, value="orders").pack(pady=5)  
 tk.Radiobutton(self.window, text="Производство", variable=self.report\_type, value="production").pack(pady=5)  
 tk.Radiobutton(self.window, text="Материалы", variable=self.report\_type, value="materials").pack(pady=5)  
 tk.Radiobutton(self.window, text="Готовая продукция", variable=self.report\_type, value="products").pack(pady=5)  
  
 tk.Button(self.window, text="Сформировать отчет", command=self.generate\_report).pack(pady=20)  
  
 def generate\_report(self):  
 report\_type = self.report\_type.get()  
 if report\_type == "orders":  
 self.generate\_orders\_report()  
 elif report\_type == "production":  
 self.generate\_production\_report()  
 elif report\_type == "materials":  
 self.generate\_materials\_report()  
 elif report\_type == "products":  
 self.generate\_products\_report()  
  
 def generate\_orders\_report(self):  
 report\_window = tk.Toplevel(self.window)  
 report\_window.title("Отчет по заказам")  
 report\_window.geometry("800x600")  
  
 tree = ttk.Treeview(report\_window, columns=(  
 "order\_id", "full\_name", "order\_date", "period\_of\_execution", "order\_status"), show="headings")  
  
 tree.heading("order\_id", text="Номер заказа")  
 tree.heading("full\_name", text="Пользователь")  
 tree.heading("order\_date", text="Дата заказа")  
 tree.heading("period\_of\_execution", text="Дата получения")  
 tree.heading("order\_status", text="Статус")  
  
 tree.column("order\_id", width=100)  
 tree.column("full\_name", width=150)  
 tree.column("order\_date", width=150)  
 tree.column("period\_of\_execution", width=150)  
 tree.column("order\_status", width=150)  
  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 scrollbar = ttk.Scrollbar(report\_window, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 orders = self.db.execute\_query("SELECT \  
 Orders.order\_id, \  
 Users.full\_name, \  
 Orders.order\_date, \  
 Orders.period\_of\_execution, \  
 Orders.order\_status \  
 FROM Orders \  
 JOIN Users ON Users.user\_id = Orders.user\_id")  
 for order in orders:  
 tree.insert("", "end", values=order)  
  
 def generate\_production\_report(self):  
 report\_window = tk.Toplevel(self.window)  
 report\_window.title("Отчет по производству")  
 report\_window.geometry("800x600")  
  
 tree = ttk.Treeview(report\_window, columns=(  
 "target\_id", "order\_id", "performer\_name", "period\_of\_execution", "target\_status"), show="headings")  
  
 tree.heading("target\_id", text="Номер задания")  
 tree.heading("order\_id", text="Номер заказа")  
 tree.heading("performer\_name", text="Исполнитель")  
 tree.heading("period\_of\_execution", text="Дата выполнения")  
 tree.heading("target\_status", text="Статус")  
  
 tree.column("target\_id", width=100)  
 tree.column("order\_id", width=100)  
 tree.column("performer\_name", width=150)  
 tree.column("period\_of\_execution", width=150)  
 tree.column("target\_status", width=150)  
  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 scrollbar = ttk.Scrollbar(report\_window, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 targets = self.db.execute\_query("SELECT \  
 ProductionTargets.target\_id, \  
 ProductionTargets.order\_id, \  
 Users.full\_name AS performer\_name, \  
 ProductionTargets.period\_of\_execution, \  
 ProductionTargets.target\_status \  
 FROM ProductionTargets \  
 JOIN Users ON Users.user\_id = ProductionTargets.performer\_id")  
 for target in targets:  
 tree.insert("", "end", values=target)  
  
 def generate\_materials\_report(self):  
 report\_window = tk.Toplevel(self.window)  
 report\_window.title("Отчет по материалам")  
 report\_window.geometry("800x600")  
  
 tree = ttk.Treeview(report\_window, columns=(  
 "material\_id", "material\_name", "quantity"), show="headings")  
  
 tree.heading("material\_id", text="ID материала")  
 tree.heading("material\_name", text="Название материала")  
 tree.heading("quantity", text="Количество")  
  
 tree.column("material\_id", width=100)  
 tree.column("material\_name", width=300)  
 tree.column("quantity", width=100)  
  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 scrollbar = ttk.Scrollbar(report\_window, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 materials = self.db.execute\_query("SELECT \  
 Materials.material\_id, \  
 Materials.material\_name, \  
 Warehouse.quantity \  
 FROM Materials \  
 JOIN Warehouse ON Materials.material\_id = Warehouse.product\_id")  
 for material in materials:  
 tree.insert("", "end", values=material)  
  
 def generate\_products\_report(self):  
 report\_window = tk.Toplevel(self.window)  
 report\_window.title("Отчет по готовой продукции")  
 report\_window.geometry("800x600")  
  
 tree = ttk.Treeview(report\_window, columns=(  
 "product\_id", "model\_name", "type\_name", "color", "size", "quantity"), show="headings")  
  
 tree.heading("product\_id", text="ID продукта")  
 tree.heading("model\_name", text="Модель")  
 tree.heading("type\_name", text="Тип")  
 tree.heading("color", text="Цвет")  
 tree.heading("size", text="Размер")  
 tree.heading("quantity", text="Количество")  
  
 tree.column("product\_id", width=100)  
 tree.column("model\_name", width=150)  
 tree.column("type\_name", width=150)  
 tree.column("color", width=100)  
 tree.column("size", width=100)  
 tree.column("quantity", width=100)  
  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 scrollbar = ttk.Scrollbar(report\_window, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 products = self.db.execute\_query("SELECT \  
 Products.product\_id, \  
 ProductModels.model\_name, \  
 ProductTypes.type\_name, \  
 Products.color, \  
 Products.size, \  
 Warehouse.quantity \  
 FROM Products \  
 JOIN ProductModels ON Products.model\_id = ProductModels.model\_id \  
 JOIN ProductTypes ON Products.type\_id = ProductTypes.type\_id \  
 JOIN Warehouse ON Products.product\_id = Warehouse.product\_id")  
 for product in products:  
 tree.insert("", "end", values=product)

import tkinter as tk  
from tkinter import ttk, messagebox  
from database import Database  
  
class WarehouseWindow:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.db = Database()  
 self.window = tk.Toplevel(self.root)  
 self.window.title("Склад")  
 self.window.geometry("800x600")  
  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 tk.Label(self.window, text="Склад", font=("Arial", 18)).pack(pady=20)  
  
 # Создаем таблицу для отображения данных склада  
 self.tree = ttk.Treeview(self.window, columns=(  
 "product\_id", "quantity", "size", "color", "model\_name", "type\_name", "material\_name"), show="headings")  
  
 # Определяем заголовки столбцов  
 self.tree.heading("product\_id", text="ID товара")  
 self.tree.heading("quantity", text="Количество")  
 self.tree.heading("size", text="Размер")  
 self.tree.heading("color", text="Цвет")  
 self.tree.heading("model\_name", text="Модель")  
 self.tree.heading("type\_name", text="Тип")  
 self.tree.heading("material\_name", text="Материал")  
  
 # Устанавливаем ширину столбцов  
 self.tree.column("product\_id", width=100)  
 self.tree.column("quantity", width=100)  
 self.tree.column("size", width=100)  
 self.tree.column("color", width=100)  
 self.tree.column("model\_name", width=100)  
 self.tree.column("type\_name", width=100)  
 self.tree.column("material\_name", width=100)  
  
 self.tree.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем полосу прокрутки  
 scrollbar = ttk.Scrollbar(self.window, orient="vertical", command=self.tree.yview)  
 self.tree.configure(yscrollcommand=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.load\_warehouse()  
  
 tk.Button(self.window, text="Добавить материал/продукцию", command=self.add\_item).pack(pady=10)  
 tk.Button(self.window, text="Редактировать", command=self.edit\_item).pack(pady=10)  
 tk.Button(self.window, text="Удалить", command=self.delete\_item).pack(pady=10)  
  
 def load\_warehouse(self):  
 # Очищаем таблицу  
 for i in self.tree.get\_children():  
 self.tree.delete(i)  
  
 # Загружаем данные склада из базы данных  
 items = self.db.execute\_query("SELECT \  
 Warehouse.product\_id, \  
 Warehouse.quantity, \  
 Products.size, \  
 Products.color, \  
 ProductModels.model\_name, \  
 ProductTypes.type\_name, \  
 Materials.material\_name \  
 FROM Warehouse \  
 JOIN Products ON Warehouse.product\_id = Products.product\_id \  
 JOIN ProductModels ON Products.model\_id = ProductModels.model\_id \  
 JOIN ProductTypes ON Products.type\_id = ProductTypes.type\_id \  
 JOIN Materials ON Products.material\_id = Materials.material\_id")  
 for item in items:  
 self.tree.insert("", "end", values=item)  
  
 def add\_item(self):  
 # Создаем диалоговое окно для добавления материала/продукции  
 add\_window = tk.Toplevel(self.window)  
 add\_window.title("Добавить материал/продукцию")  
 add\_window.geometry("400x300")  
  
 tk.Label(add\_window, text="ID товара:").pack(pady=5)  
 product\_id\_entry = tk.Entry(add\_window)  
 product\_id\_entry.pack(pady=5)  
  
 tk.Label(add\_window, text="Количество:").pack(pady=5)  
 quantity\_entry = tk.Entry(add\_window)  
 quantity\_entry.pack(pady=5)  
  
 def save\_item():  
 product\_id = product\_id\_entry.get()  
 quantity = quantity\_entry.get()  
  
 if not product\_id or not quantity:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 self.db.execute\_query(  
 "INSERT INTO Warehouse (product\_id, quantity) VALUES (%s, %s)",  
 (product\_id, quantity), fetch=False)  
 add\_window.destroy()  
 self.load\_warehouse()  
  
 tk.Button(add\_window, text="Сохранить", command=save\_item).pack(pady=10)  
  
 def edit\_item(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите товар для редактирования")  
 return  
  
 product\_id = self.tree.item(selected\_item)['values'][0]  
 quantity = self.tree.item(selected\_item)['values'][1]  
  
 # Создаем диалоговое окно для редактирования материала/продукции  
 edit\_window = tk.Toplevel(self.window)  
 edit\_window.title("Редактировать материал/продукцию")  
 edit\_window.geometry("400x300")  
  
 tk.Label(edit\_window, text="ID товара:").pack(pady=5)  
 product\_id\_entry = tk.Entry(edit\_window)  
 product\_id\_entry.insert(0, product\_id)  
 product\_id\_entry.pack(pady=5)  
  
 tk.Label(edit\_window, text="Количество:").pack(pady=5)  
 quantity\_entry = tk.Entry(edit\_window)  
 quantity\_entry.insert(0, quantity)  
 quantity\_entry.pack(pady=5)  
  
 def update\_item():  
 new\_product\_id = product\_id\_entry.get()  
 new\_quantity = quantity\_entry.get()  
  
 if not new\_product\_id or not new\_quantity:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 self.db.execute\_query(  
 "UPDATE Warehouse SET quantity = %s WHERE product\_id = %s",  
 (new\_quantity, new\_product\_id), fetch=False)  
 edit\_window.destroy()  
 self.load\_warehouse()  
  
 tk.Button(edit\_window, text="Сохранить", command=update\_item).pack(pady=10)  
  
 def delete\_item(self):  
 selected\_item = self.tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите товар для удаления")  
 return  
  
 product\_id = self.tree.item(selected\_item)['values'][0]  
 confirm = messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить товар №{product\_id}?")  
 if confirm:  
 self.db.execute\_query("DELETE FROM Warehouse WHERE product\_id = %s", (product\_id,), fetch=False)  
 self.load\_warehouse()

## unit-тесты

import unittest  
from unittest.mock import patch, MagicMock  
from tkinter import messagebox  
from ui.login\_window import LoginWindow  
from ui.main\_window import MainWindow  
import re  
  
class TestLoginWindow(unittest.TestCase):  
 @patch('database.Database.execute\_query')  
 @patch('tkinter.messagebox.showinfo')  
 @patch('tkinter.Tk.destroy')  
 @patch('ui.login\_window.LoginWindow.open\_main\_window') # Замокать метод, который открывает главное окно  
 def test\_login\_success(self, mock\_open\_main\_window, mock\_destroy, mock\_showinfo, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [(1, 'Администратор', 'Иван Иванов', 'admin', 'password', '1234567890')]  
  
 # Создание экземпляра LoginWindow  
 login\_window = LoginWindow(MagicMock())  
  
 # Мокирование полей ввода  
 login\_window.entry\_login = MagicMock()  
 login\_window.entry\_password = MagicMock()  
  
 # Установка side\_effect для метода get  
 login\_window.entry\_login.get.side\_effect = lambda: 'admin'  
 login\_window.entry\_password.get.side\_effect = lambda: 'password'  
  
 # Вызов метода login  
 login\_window.login()  
  
 # Проверка ожидаемых результатов  
 expected\_query = re.compile(r"SELECT\s+Users\.user\_id,\s+Roles\.role\_name,\s+Users\.full\_name,\s+Users\.login,\s+Users\.password,\s+Users\.phone\s+FROM\s+Users\s+JOIN\s+Roles\s+ON\s+Users\.role\_id\s+=\s+Roles\.role\_id\s+WHERE\s+login\s+=\s+'admin'\s+and\s+password\s+=\s+'password'")  
 mock\_execute\_query.assert\_called\_once()  
 self.assertRegex(mock\_execute\_query.call\_args[0][0], expected\_query)  
 mock\_showinfo.assert\_called\_once\_with("Успех", "Авторизация прошла успешно!")  
 # mock\_destroy.assert\_called\_once() # Проверка, что метод destroy был вызван  
 mock\_open\_main\_window.assert\_called\_once() # Проверка, что метод открытия главного окна был вызван  
  
 @patch('database.Database.execute\_query')  
 @patch('tkinter.messagebox.showerror')  
 def test\_login\_failure(self, mock\_showerror, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = []  
  
 # Создание экземпляра LoginWindow  
 login\_window = LoginWindow(MagicMock())  
  
 # Мокирование полей ввода  
 login\_window.entry\_login = MagicMock()  
 login\_window.entry\_password = MagicMock()  
  
 # Установка side\_effect для метода get  
 login\_window.entry\_login.get.side\_effect = lambda: 'invalid\_user'  
 login\_window.entry\_password.get.side\_effect = lambda: 'wrong\_password'  
  
 # Вызов метода login  
 login\_window.login()  
  
 # Проверка ожидаемых результатов  
 expected\_query = re.compile(r"SELECT\s+Users\.user\_id,\s+Roles\.role\_name,\s+Users\.full\_name,\s+Users\.login,\s+Users\.password,\s+Users\.phone\s+FROM\s+Users\s+JOIN\s+Roles\s+ON\s+Users\.role\_id\s+=\s+Roles\.role\_id\s+WHERE\s+login\s+=\s+'invalid\_user'\s+and\s+password\s+=\s+'wrong\_password'")  
 mock\_execute\_query.assert\_called\_once()  
 self.assertRegex(mock\_execute\_query.call\_args[0][0], expected\_query)  
 mock\_showerror.assert\_called\_once\_with("Ошибка", "Неверный логин или пароль")  
  
  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

import unittest  
from unittest.mock import patch, MagicMock  
from tkinter import messagebox  
from ui.login\_window import LoginWindow  
from ui.main\_window import MainWindow  
import re  
  
class TestMainWindow(unittest.TestCase):  
 @patch('ui.main\_window.OrdersWindow')  
 @patch('ui.main\_window.MainWindow.create\_widgets')  
 def test\_open\_orders(self, mock\_create\_widgets, mock\_orders\_window):  
 # Создание экземпляра MainWindow  
 main\_window = MainWindow(MagicMock(), [(1, 'Администратор', 'Иван Иванов', 'admin', 'password', '1234567890')])  
  
 # Вызов метода open\_orders  
 main\_window.open\_orders()  
  
 # Проверка, что OrdersWindow был вызван с правильными аргументами  
 mock\_orders\_window.assert\_called\_once\_with(main\_window.root, main\_window.user\_info)  
  
 @patch('database.Database.execute\_query')  
 @patch('tkinter.messagebox.showerror')  
 @patch('tkinter.Toplevel')  
 @patch('ui.main\_window.MainWindow.create\_widgets')  
 def test\_track\_order(self, mock\_create\_widgets, mock\_toplevel, mock\_showerror, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, '2023-10-01', 'В обработке', 'ул. Ленина, 1', '1234567890', 'Иван Иванов', 'Продукт 1', '123456', 2, 100.0)  
 ]  
  
 # Создание экземпляра MainWindow  
 main\_window = MainWindow(MagicMock(), [(1, 'Администратор', 'Иван Иванов', 'admin', 'password', '1234567890')])  
  
 # Мокирование ввода данных  
 main\_window.order\_entry = MagicMock()  
 main\_window.order\_entry.get.return\_value = '1'  
  
 # Вызов метода track\_order  
 main\_window.track\_order()  
  
 # Проверка, что execute\_query был вызван с правильными аргументами  
 expected\_query = re.compile(r"""  
 SELECT\s+Orders\.order\_date,\s+Orders\.period\_of\_execution,\s+Orders\.order\_status,\s+  
 OrderLines\.quantity,\s+OrderLines\.total\_price,\s+Products\.size,\s+Products\.color,\s+  
 ProductModels\.model\_name,\s+ProductTypes\.type\_name,\s+Materials\.material\_name\s+  
 FROM\s+Orders\s+  
 JOIN\s+OrderLines\s+ON\s+Orders\.order\_id\s+=\s+OrderLines\.order\_id\s+  
 JOIN\s+Products\s+ON\s+OrderLines\.product\_id\s+=\s+Products\.product\_id\s+  
 JOIN\s+ProductModels\s+ON\s+Products\.model\_id\s+=\s+ProductModels\.model\_id\s+  
 JOIN\s+ProductTypes\s+ON\s+Products\.type\_id\s+=\s+ProductTypes\.type\_id\s+  
 JOIN\s+Materials\s+ON\s+Products\.material\_id\s+=\s+Materials\.material\_id\s+  
 WHERE\s+Orders\.order\_id\s+=\s+%s;  
 """, re.VERBOSE)  
 mock\_execute\_query.assert\_called\_once()  
 self.assertRegex(mock\_execute\_query.call\_args[0][0], expected\_query)  
  
 # Проверка, что showerror не был вызван  
 mock\_showerror.assert\_not\_called()  
  
 # Проверка, что Toplevel был вызван  
 mock\_toplevel.assert\_called\_once()  
  
 @patch('database.Database.execute\_query')  
 @patch('tkinter.messagebox.showerror')  
 @patch('ui.main\_window.MainWindow.create\_widgets')  
 def test\_track\_order\_not\_found(self, mock\_create\_widgets, mock\_showerror, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = []  
  
 # Создание экземпляра MainWindow  
 main\_window = MainWindow(MagicMock(), [(1, 'Администратор', 'Иван Иванов', 'admin', 'password', '1234567890')])  
  
 # Мокирование ввода данных  
 main\_window.order\_entry = MagicMock()  
 main\_window.order\_entry.get.return\_value = '1'  
  
 # Вызов метода track\_order  
 main\_window.track\_order()  
  
 # Проверка, что execute\_query был вызван с правильными аргументами  
 expected\_query = re.compile(r"""  
 SELECT\s+Orders\.order\_date,\s+Orders\.period\_of\_execution,\s+Orders\.order\_status,\s+  
 OrderLines\.quantity,\s+OrderLines\.total\_price,\s+Products\.size,\s+Products\.color,\s+  
 ProductModels\.model\_name,\s+ProductTypes\.type\_name,\s+Materials\.material\_name\s+  
 FROM\s+Orders\s+  
 JOIN\s+OrderLines\s+ON\s+Orders\.order\_id\s+=\s+OrderLines\.order\_id\s+  
 JOIN\s+Products\s+ON\s+OrderLines\.product\_id\s+=\s+Products\.product\_id\s+  
 JOIN\s+ProductModels\s+ON\s+Products\.model\_id\s+=\s+ProductModels\.model\_id\s+  
 JOIN\s+ProductTypes\s+ON\s+Products\.type\_id\s+=\s+ProductTypes\.type\_id\s+  
 JOIN\s+Materials\s+ON\s+Products\.material\_id\s+=\s+Materials\.material\_id\s+  
 WHERE\s+Orders\.order\_id\s+=\s+%s;  
 """, re.VERBOSE)  
 mock\_execute\_query.assert\_called\_once()  
 self.assertRegex(mock\_execute\_query.call\_args[0][0], expected\_query)  
  
 # Проверка, что showerror был вызван с правильными аргументами  
 mock\_showerror.assert\_called\_once\_with("Ошибка", "Заказ не найден")  
  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

import unittest  
from unittest.mock import patch, MagicMock  
from tkinter import Tk  
from ui.orders\_window import OrdersWindow  
  
class TestOrdersWindow(unittest.TestCase):  
 def setUp(self):  
 self.root = Tk()  
 self.user\_info = (1, 'Admin', 'Admin User', 'admin', 'password', '1234567890')  
 self.orders\_window = OrdersWindow(self.root, self.user\_info)  
  
 def tearDown(self):  
 self.root.destroy()  
  
 @patch('orders\_window.Database.execute\_query')  
 def test\_load\_orders(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, 'User1', '2023-10-01 12:00:00', '2023-10-10 12:00:00', 'Completed'),  
 (2, 'User2', '2023-10-02 12:00:00', '2023-10-11 12:00:00', 'Pending')  
 ]  
  
 # Вызов метода, который тестируем  
 self.orders\_window.load\_orders()  
  
 # Проверка, что данные были загружены в Treeview  
 self.assertEqual(len(self.orders\_window.tree.get\_children()), 2)  
  
 @patch('orders\_window.Database.execute\_query')  
 def test\_add\_order(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = None  
  
 # Вызов метода, который тестируем  
 self.orders\_window.add\_order()  
  
 # Проверка, что данные были добавлены в базу данных  
 mock\_execute\_query.assert\_called\_once()  
  
 @patch('orders\_window.Database.execute\_query')  
 def test\_edit\_order(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = None  
  
 # Вызов метода, который тестируем  
 self.orders\_window.edit\_order()  
  
 # Проверка, что данные были обновлены в базе данных  
 mock\_execute\_query.assert\_called\_once()  
  
 @patch('orders\_window.Database.execute\_query')  
 def test\_delete\_order(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = None  
  
 # Вызов метода, который тестируем  
 self.orders\_window.delete\_order()  
  
 # Проверка, что данные были удалены из базы данных  
 mock\_execute\_query.assert\_called\_once()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

import unittest  
from unittest.mock import patch, MagicMock  
from tkinter import Tk  
from tkinter import ttk # Добавьте этот импорт  
from ui.reports\_window import ReportsWindow  
  
class TestReportsWindow(unittest.TestCase):  
  
 def setUp(self):  
 self.root = Tk()  
 self.reports\_window = ReportsWindow(self.root)  
  
 def tearDown(self):  
 self.root.destroy()  
  
 @patch('ui.reports\_window.Database.execute\_query')  
 def test\_generate\_orders\_report(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, 'User1', '2023-10-01 12:00:00', '2023-10-10 12:00:00', 'Completed'),  
 (2, 'User2', '2023-10-02 12:00:00', '2023-10-11 12:00:00', 'Pending')  
 ]  
  
 # Вызов метода, который тестируем  
 self.reports\_window.generate\_orders\_report()  
  
 # Проверка, что данные были загружены в Treeview  
 report\_window = self.root.winfo\_children()[-1]  
 tree = None  
 for child in report\_window.winfo\_children():  
 if isinstance(child, ttk.Treeview):  
 tree = child  
 break  
 self.assertIsNotNone(tree, "Treeview не найден в окне отчета")  
 self.assertEqual(len(tree.get\_children()), 2)  
  
 @patch('ui.reports\_window.Database.execute\_query')  
 def test\_generate\_production\_report(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, 1, 'Performer1', '2023-10-10 12:00:00', 'Completed'),  
 (2, 2, 'Performer2', '2023-10-11 12:00:00', 'Pending')  
 ]  
  
 # Вызов метода, который тестируем  
 self.reports\_window.generate\_production\_report()  
  
 # Проверка, что данные были загружены в Treeview  
 report\_window = self.root.winfo\_children()[-1]  
 tree = None  
 for child in report\_window.winfo\_children():  
 if isinstance(child, ttk.Treeview):  
 tree = child  
 break  
 self.assertIsNotNone(tree, "Treeview не найден в окне отчета")  
 self.assertEqual(len(tree.get\_children()), 2)  
  
 @patch('ui.reports\_window.Database.execute\_query')  
 def test\_generate\_materials\_report(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, 'Material1', 100),  
 (2, 'Material2', 200)  
 ]  
  
 # Вызов метода, который тестируем  
 self.reports\_window.generate\_materials\_report()  
  
 # Проверка, что данные были загружены в Treeview  
 report\_window = self.root.winfo\_children()[-1]  
 tree = None  
 for child in report\_window.winfo\_children():  
 if isinstance(child, ttk.Treeview):  
 tree = child  
 break  
 self.assertIsNotNone(tree, "Treeview не найден в окне отчета")  
 self.assertEqual(len(tree.get\_children()), 2)  
  
 @patch('ui.reports\_window.Database.execute\_query')  
 def test\_generate\_products\_report(self, mock\_execute\_query):  
 # Подготовка данных для теста  
 mock\_execute\_query.return\_value = [  
 (1, 'Model1', 'Type1', 'Red', 'M', 50),  
 (2, 'Model2', 'Type2', 'Blue', 'L', 100)  
 ]  
  
 # Вызов метода, который тестируем  
 self.reports\_window.generate\_products\_report()  
  
 # Проверка, что данные были загружены в Treeview  
 report\_window = self.root.winfo\_children()[-1]  
 tree = None  
 for child in report\_window.winfo\_children():  
 if isinstance(child, ttk.Treeview):  
 tree = child  
 break  
 self.assertIsNotNone(tree, "Treeview не найден в окне отчета")  
 self.assertEqual(len(tree.get\_children()), 2)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**АТТЕСТАЦИОННЫЙ ЛИСТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ УП.02.01 (ПО ПРОФИЛЮ СПЕЦИАЛЬНОСТИ)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность 09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент 4 курса 42919/9 группы

Михеев Александр Сергеевич

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, пр. Энгельса, 23

(наименование и адрес организации)

Период прохождения практики

с «09» сентября 2024 г. по «21» сентября 2024 г.

**Виды и качество выполнения работ**

|  |  |  |
| --- | --- | --- |
| **Виды выполненных работ обучающимся во время практики** | **Объем работ, час.** | **Качество выполнения работ (оценка по пятибалльной системе)** |
| Участие в выработке требований к программному обеспечению | 9 |  |
| Стадии проектирования программного обеспечения | 21 |  |
| Разработка модулей программного обеспечения | 24 |  |
| Тестирование программных модулей и их интеграции | 9 |  |
| Разработка программной документации и стандарты кодирования | 9 |  |

**Характеристика учебной/профессиональной деятельности обучающегося во время учебной практики (по профилю специальности):**

Общие и профессиональные компетенции, предусмотренные программой практики, освоены **/** не освоены.

(нужное подчеркнуть)

Итоговая оценка по практике

Руководитель практики Хисамутдинова А.С.

(Ф.И.О.) (подпись)

Дата «21» сентября 2024 г.