**КУРСОВАЯ РАБОТА**

по дисциплине «Программная инженерия»

Выполнил студент Ефремов А.Д.

Руководитель Самочадин А.В.

10.12.2024г.

# Реферат

**Название проекта:** Персонал;

**Аннотация проекта:** ПО для взаимодействия клиентов, арендодателя и работников (клининг, сантехники и т.д.);

**Цели и задачи проекта:** Создание приложения, облегчающее взаимодействия между клиентами и арендодателями, а также между работниками и арендодателями. Оно предоставляет инструкции для гостя, общение, управление документами, онлайн-платежи и даже вызов специалистов. Цель - сделать процесс аренды прозрачным, безопасным и удобным для всех.

**Основные задачи:**

-Создание удобной платформы;

-Автоматизация управления сотрудниками;

-Дополнение ПО различными функциями;

**Ожидаемые результаты:**

-Рабочее ПО с удобным интерфейсом;

-Система уведомлений через PUSH;

-Интеграция с наиболее популярными сервисами ведения бизнеса посуточной аренды (RealtyCalendar);

-Возможность расширения функциональности приложения;

**Области применения результатов:**

-Платформы сдачи квартир (AirBNB, Островок и т.д.)

-Малый бизнес;

**Потенциальные потребители продукта:**

-Арендаторы;

-Арендодатели;

-Специалисты, предлагающие свои услуги по уборке/ремонту;

# Содержание

[Реферат 1](#_Toc184765929)

[Содержание 2](#_Toc184765930)

[Введение 3](#_Toc184765931)

[Анализ требований и уточнение спецификаций 4](#_Toc184765932)

[Выбор и описание методологии разработки 4](#_Toc184765933)

[Разработка вариантов использования 6](#_Toc184765934)

[Пользовательские требования к ПО, составленные на основе сценариев вариантов использования 9](#_Toc184765935)

[Выбор и описание технологического стека 9](#_Toc184765936)

[Детальные требования 11](#_Toc184765937)

[Перечень работ проекта (WBS) 13](#_Toc184765938)

[Оценка трудозатрат на выполнение проекта 14](#_Toc184765939)

[Перечень работ по проекту, выполненных автором курсовой работы 18](#_Toc184765940)

[Проектирование структуры и компонентов программного продукта 21](#_Toc184765941)

[Архитектура разрабатываемого ПО 21](#_Toc184765942)

[Модель предметной области 33](#_Toc184765943)

[Описание интерфейса пользователя 34](#_Toc184765944)

[Описание схемы БД 35](#_Toc184765945)

[Проектирование компонентов 38](#_Toc184765946)

[Описание процесса кодирования 42](#_Toc184765947)

[Выбор стратегии тестирования и разработка тестов 46](#_Toc184765948)

[Стратегия тестирования 46](#_Toc184765949)

[Описание процесса модульного и интеграционного тестирования 48](#_Toc184765950)

[Тест-кейсы 52](#_Toc184765951)

[Заключение 62](#_Toc184765952)

[Литература 63](#_Toc184765953)

[Приложения 64](#_Toc184765954)

[Приложение А. Руководство пользователя 64](#_Toc184765955)

[Приложение Б. Сoding standard 67](#_Toc184765956)

[Приложение В. Фрагменты текстов программ 70](#_Toc184765957)

# Введение

В последние десятилетия мир технологий и инноваций активно меняет различные сферы нашей жизни, включая и сферу предоставления профессиональных услуг. С развитием информационных технологий и популяризацией интернета, создание удобных платформ для взаимодействия между специалистами и их потенциальными заказчиками стало важной задачей для многих отраслей.

В этой связи появляется потребность в сервисах, которые могут обеспечить простой и удобный способ поиска профессионалов для выполнения различных задач. Эти задачи могут варьироваться от ремонтных работ и установки оборудования до предоставления консультационных услуг и обслуживания бытовой техники. Традиционные методы поиска специалистов, такие как объявления в газетах или реклама на досках объявлений, становятся все менее актуальными и не всегда позволяют найти квалифицированных профессионалов с хорошей репутацией и необходимыми навыками. Именно в ответ на эту потребность возникает необходимость в платформе, которая бы оптимизировала этот процесс и сделала его максимально удобным для обеих сторон.

Наш проект представляет собой такую платформу, которая связывает заказчиков и специалистов для предоставления различных услуг. В рамках этого проекта пользователи могут легко найти квалифицированных специалистов для выполнения задач, связанных с их домом или квартирой, будь то ремонтные работы, установка бытовой техники, консультации по вопросам энергоэффективности или другие услуги. Платформа предоставляет удобный интерфейс для подачи заявок, поиска и выбора исполнителей, а также для отслеживания статуса выполнения услуг.

Основное внимание в проекте уделено удобству пользователя, безопасности взаимодействия и обеспечению прозрачности. Все специалисты, представленные на платформе, проходят процедуру проверки и регистрации, что позволяет заказчикам быть уверенными в их квалификации

Для реализации платформы использованы современные технологии программирования на языке Java и библиотеках, таких как Swing. Обеспечена интеграция с базами данных для хранения информации, а также для защиты данных пользователей с использованием современных методов безопасности, таких как хеширование паролей с помощью алгоритма BCrypt.

Таким образом, проект представляет собой удобную и безопасную платформу для поиска и предоставления профессиональных услуг, предназначенную для удовлетворения потребностей пользователей в квалифицированных специалистах для работы с жильем и домашними задачами.

# Анализ требований и уточнение спецификаций

## Выбор и описание методологии разработки

Модель жизненного цикла: Водопад (Waterfall)

Для данного проекта используется модель жизненного цикла разработки «Водопад»:

1. Анализ требований
2. Проектирование системы
3. Реализация (разработка)
4. Тестирование
5. Внедрение
6. Эксплуатация и сопровождение

Особенностью данной модели является то, что каждый этап должен быть завершен перед началом следующего.

Описание процесса

1. Анализ требований: Сбор и формализация всех требований к системе. На данном этапе создается документ, в котором описываются функциональные и нефункциональные требования к приложению для управления бизнесом посуточной аренды.
   * Тип итераций: Линейная фаза, без возврата на предыдущие этапы.
   * Продолжительность: 2 недели.
2. Проектирование системы: Разработка архитектурных решений и дизайна системы. Описываются интерфейсы, общая структура приложения, планируется база данных, объектная модель.
   * Тип итераций: Линейная фаза, переход на следующий этап только после полного завершения.
   * Продолжительность: 3 недели.
3. Реализация (разработка): Кодирование системы в соответствии с технической документацией. Программные модули создаются и интегрируются в единую систему.
   * Тип итераций: Линейная фаза с возможностью делегирования задач между разработчиками.
   * Продолжительность: 4 недели.
4. Тестирование: Проверка системы на соответствие требованиям, устранение ошибок и улучшение качества.
   * Тип итераций: Линейная фаза с возможностью проведения нескольких циклов тестирования.
   * Продолжительность: 2 недели.
5. Внедрение: Окончательная установка системы в рабочее окружение.
   * Тип итераций: Линейная фаза с единовременным разворачиванием.
   * Продолжительность: 1 неделя.
6. Эксплуатация и сопровождение: Техническая поддержка системы и исправление возникающих проблем в процессе эксплуатации.
   * Тип итераций: Непрерывный процесс по мере необходимости.
   * Продолжительность: Неопределённая, зависит от сроков эксплуатации.

Рабочие продукты

В ходе выполнения проекта создаются следующие рабочие продукты:

1. Документ с требованиями к системе – описывает все функциональные и нефункциональные требования к приложению.
2. План релизов – содержит информацию о версии системы, датах выпуска релизов и ключевых функциях каждого релиза.
3. Отчеты о состоянии работ по проекту – регулярные отчеты о текущем статусе проекта, прогрессе и выполненных задачах.
4. Документ, описывающий дизайн интерфейсов – включает макеты экранов, взаимодействие пользователя с системой, принципы UX/UI.
5. Общая объектная модель системы – диаграмма классов, связи объектов и взаимодействие между ними.
6. Спецификации – технические описания всех разработанных модулей системы, их функциональность и взаимодействие.

Стандарты

Для выполнения проекта используются следующие стандарты:

1. Стандарты кодирования – требования к написанию кода, включая форматирование, именование переменных, использование комментариев, структуры классов и методов.
2. Стандарты построения пользовательских интерфейсов – рекомендации и принципы проектирования UI, учитывающие удобство работы пользователей с приложением.
3. Стандарты тестирования – методы и стратегии тестирования, которые должны применяться для проверки системы.

## Разработка вариантов использования

**Перечень действующих лиц (Actors):**

1. **Менеджер (Manager)**:
   * Управляет заказами и взаимодействует с услугами и специалистами.
2. **Работник/Специалист (Worker)**:
   * Выполняет заказы и управляет предоставляемыми услугами.
3. **Администратор (Admin)**:
   * Управляет системой: пользователями, услугами, заказами, правами доступа.

Изображение выглядит как рисунок, зарисовка, текст, диаграмма

Автоматически созданное описание

**Сценарии вариантов использования (Use Case)**

**1. Вход в систему (Log in)**

* **Действующие лица:** Пользователь
* **Описание:** Пользователь вводит логин и пароль для входа в систему.
* **Шаги:**
  1. Пользователь открывает форму входа.
  2. Вводит логин и пароль.
  3. Система проверяет учетные данные.
  4. В случае успешной проверки пользователь получает доступ к своему аккаунту.

**2. Регистрация нового пользователя (Sign up)**

* **Действующие лица:** Пользователь.
* **Описание:** Пользователь создает учетную запись для работы в системе.
* **Шаги:**
  1. Пользователь открывает форму регистрации.
  2. Вводит личные данные (имя, email, пароль и т. д.).
  3. Система сохраняет данные в базе и создает учетную запись.
* **Расширения:**

2.а Данные оказались некорректными

2.а2 Система вывела ошибку при регистрации

**3. Просмотр списка услуг (View list of services)**

* **Действующие лица:** Менеджер, Администратор.
* **Описание:** Действующее лицо просматривает доступный список услуг.
* **Шаги:**
  1. Пользователь открывает раздел "Список услуг".
  2. Система отображает список всех доступных услуг.

**4. Создание нового заказа (Order a new service)**

* **Действующие лица:** Пользователь.
* **Описание:** Пользователь оформляет заказ на услугу.
* **Шаги:**
  1. Пользователь выбирает услугу из списка.
  2. Указывает данные заказа (описание, сроки и т. д.).
  3. Система сохраняет заказ как "не выполненный".

**5. Просмотр невыполненных заказов (View unfulfilled orders)**

* **Действующие лица:** Менеджер, Администратор.
* **Описание:** Менеджер или администратор просматривает заказы, которые еще не выполнены.
* **Шаги:**
  1. Пользователь открывает раздел "Невыполненные заказы".
  2. Система отображает список заказов со статусом "не выполнен".

**6. Изменение данных услуги (Change services data)**

* **Действующие лица:** Работник, Администратор.
* **Описание:** Работник или администратор изменяет информацию об услуге.
* **Шаги:**
  1. Пользователь выбирает услугу из списка.
  2. Вносит изменения в название, описание или стоимость услуги.
  3. Система сохраняет изменения в базе данных.

**7. Назначение статуса пользователю (Give administrator/worker status)**

* **Действующие лица:** Администратор.
* **Описание:** Администратор изменяет статус пользователя, назначая его администратором или специалистом.
* **Шаги:**
  1. Администратор открывает список пользователей.
  2. Выбирает пользователя для изменения статуса.
  3. Указывает новый статус (администратор или специалист).
  4. Система сохраняет изменения.

**8. Выполнение заказа (Send the completed order ID)**

* **Действующие лица:** Работник.
* **Описание:** Работник завершает выполнение заказа.
* **Шаги:**
  1. Работник открывает список доступных заказов.
  2. Отмечает заказ как "выполненный".
  3. Система обновляет статус заказа и фиксирует дату завершения.

**9. Управление пользователями (Make action with list of users)**

* **Действующие лица:** Администратор.
* **Описание:** Администратор управляет учетными записями пользователей.
* **Шаги:**
  1. Администратор открывает список пользователей.
  2. Выбирает пользователя.
  3. Может выполнить следующие действия:
     + Изменить данные.
     + Удалить учетную запись.
     + Назначить роль.
  4. Система сохраняет изменения.
* **Расширения:**

3.а Данные оказались некорректными

3.а2 Система вывела ошибку

**10. Просмотр выполненных заказов (View completed orders)**

* **Действующие лица:** Менеджер, Администратор.
* **Описание:** Менеджер или администратор просматривает список завершенных заказов.
* **Шаги:**
  1. Пользователь открывает раздел "Выполненные заказы".
  2. Система отображает список заказов со статусом "выполнен".

## Пользовательские требования к ПО, составленные на основе сценариев вариантов использования

**1. Общие требования**

1.1. Система должна обеспечивать авторизацию пользователей через форму входа.  
1.2. Система должна предоставлять возможность регистрации для новых пользователей.

**2. Требования для роли "Администратор"**

2.1. Администратор должен иметь доступ к следующим действиям:

* Просмотр списка всех пользователей.
* Просмотр списка работников (специалистов).
* Просмотр списка администраторов.
* Просмотр списка услуг.
* Просмотр списка всех заказов (выполненных и невыполненных). 2.2. Администратор должен иметь возможность:
* Изменять данные услуг.
* Удалять работников и администраторов из базы данных.
* Изменять статус пользователей:
  + Назначать пользователя администратором.
  + Назначать пользователя работником (специалистом).

**3. Требования для роли "Работник (специалист)"**

3.1. Работник должен иметь доступ к следующим действиям:

* Просмотр доступных для выполнения заказов.
* Просмотр своих активных заказов.
* Отправка завершённого заказа. 3.2. Работник должен иметь возможность изменять список предоставляемых услуг.

**4. Требования для роли "Пользователь"**

4.1. Пользователь должен иметь доступ к следующему:

* Просмотр списка услуг.
* Оформление нового заказа на услугу.

**5. Общие функциональные расширения**

5.1. Система должна поддерживать гибкость в изменении данных (например, обновление информации о пользователях, заказах, услугах).  
5.2. Система должна позволять просматривать детализированную информацию о заказах (выполненных и невыполненных).

## Выбор и описание технологического стека

**1. Языки программирования**

**• Java :**

* **Почему выбран**: Java является одним из самых популярных и зрелых языков программирования для разработки приложений. Он имеет широкое сообщество, множество библиотек и инструментов, а также длительную историю в разработке.
* **Тип использования:** Java будет использоваться для логики интерфейса и взаимодействия с серверными API через библиотеки и фреймворки.

**2. Фреймворки**

* **React Native**:
  + **Почему выбран**: React Native — это кроссплатформенный фреймворк. Он обеспечивает хорошую производительность и позволяет быстро создать прототип с единой кодовой базой для обеих платформ. React Native также поддерживает доступ к нативным возможностям устройства через плагины и нативные модули.
  + **Тип использования**: Разработка пользовательского интерфейса и взаимодействие с внешними сервисами.

**3. Библиотеки и инструменты для разработки**

* **React Navigation**:
  + Почему выбран: Библиотека для управления навигацией. Обеспечивает легкую настройку маршрутов и переходов между экранами, что делает интерфейс удобным для пользователя.
  + Тип использования: Управление навигацией между экранами приложения, такими как поиск клининговых компаний, заказ услуги, личный кабинет пользователя.

**4. Бэкенд и базы данных**

* **PostgreSQL**:
  + **Почему выбран**: PostgreSQL — это реляционная база данных с высокой производительностью, которая поддерживает сложные SQL-запросы и масштабируется под нагрузкой. Ее выбор обусловлен тем, что данные о пользователях, клининговых компаниях, заказах и платежах требуют реляционной структуры.
  + **Тип использования**: Хранение данных о пользователях, клининговых компаниях, заказах и платежах.

**5. Средства разработки и тестирования**

**• IntelliJ IDEA:**

* **Почему выбран:** IntelliJ IDEA — это мощная интегрированная среда разработки (IDE), которая поддерживает широкий спектр языков и технологий, включая Java, а также различные фреймворки для работы с серверными API. IDEA предоставляет улучшенное автодополнение кода, рефакторинг и отладку. Также имеет множество плагинов для работы с различными языками и платформами.
* **Тип использования**: Основной инструмент разработки для создания и отладки приложений, а также для написания серверного кода, взаимодействующего с API.

**6. Обоснование выбора технологического стека**

Выбор описанного стека обусловлен необходимостью разработки кроссплатформенного приложения с минимальными затратами времени и ресурсов. React Native позволяет использовать единую кодовую базу для разных платформ, что значительно ускоряет процесс разработки. Язык программирования Java является основой как для фронтенда, так и для бэкенда, что упрощает поддержание приложения и способствует эффективному взаимодействию между разработчиками.

PostgreSQL выбран для хранения данных благодаря своей масштабируемости и надежности.

**Заключение:**

Технологический стек для приложения по найму клининговых услуг был выбран с учетом требований к производительности, кроссплатформенности и простоты поддержки. Он позволяет разработать качественное мобильное приложение с минимальными затратами и гибкой архитектурой для будущих изменений и улучшений.

## Детальные требования

**1. Функциональные требования**

**1.1. Управление пользователями**

* **Добавление пользователя:**
  + Форма для ввода данных: логин пароль.
  + Уведомление об ошибках при вводе данных, включая неподходящий пароль.
* **Редактирование информации о пользователе:**
  + Возможность изменения данных (роль, контактная информация).
* **Удаление пользователя:**
  + Удаление записи с подтверждением действия.
* **Просмотр списка пользователей:**
  + Отображение списка всех пользователей с возможностью фильтрации по ролям.

**1.2. Управление заказами**

* **Создание нового заказа:**
  + Форма для ввода данных: услуга, описание, назначенный специалист.
* **Просмотр заказов:**
  + Список всех заказов с фильтрацией по статусу (не выполнен, выполняется, выполнен).
* **Редактирование данных заказа:**
  + Изменение описания, срока или назначенного специалиста.
* **Удаление заказа:**
  + Удаление записи заказа с подтверждением.

**1.3. Управление услугами**

* **Добавление новой услуги:**
  + Ввод данных: название, описание, стоимость.
* **Редактирование услуг:**
  + Изменение описания или стоимости услуги.
* **Удаление услуги:**
  + Удаление услуги с проверкой на отсутствие связанных активных заказов.
* **Просмотр списка услуг:**
  + Вывод списка всех доступных услуг.

**1.4. Управление статусами заказов**

* **Изменение статуса заказа:**
  + Возможность назначить статус.
* **Отчет о выполнении заказа:**
  + Специалист отправляет отчет о завершении заказа.

**1.5. Управление ролями и правами доступа**

* **Назначение ролей:**
  + Возможность изменения роли пользователя (например, назначение администратора или специалиста).
* **Ограничение доступа:**
  + Разграничение функциональности в зависимости от роли:
    - Администратор: полный доступ.
    - Менеджер: управление заказами и специалистами.
    - Специалист: выполнение назначенных заказов.

**3. Нефункциональные требования**

**3.1. Производительность**

* Обработка данных для 1000 пользователей и 5000 заказов.
* Среднее время отклика системы — не более 1 секунды.

**3.2. Удобство использования**

* Поддержка работы на ПК
* Интерфейс без дополнительных вкладок

**3.3. Масштабируемость**

* Возможность интеграции с корпоративными системами (например, CRM или ERP при некоторой доработке).

**4. Пользовательские роли**

**4.1. Администратор**

* Управление пользователями, услугами, заказами.
* Доступ к логам системы и настройкам.

**4.2. Клиент**

* Создание заказа.
* Просмотр услуг, заказов.

**4.3. Специалист**

* Просмотр назначенных заказов.
* Изменение статуса заказов, отправка отчетов о выполнении.

**6. Технические требования**

* **Язык программирования:** Java (для графического интерфейса и бэкенда).
* **База данных:** PostgreSQL для хранения данных.
* **Интерфейс:** Swing.

**7. Тестирование**

* **Юнит-тесты:** Проверка отдельных модулей (например, управления пользователями).
* **Интеграционные тесты:** Проверка взаимодействия между модулями.
* **Тестирование интерфейса:** Оценка удобства и доступности интерфейса пользователями.

## Перечень работ проекта (WBS)

1. Инициализация проекта
   1. Формирование команды
   2. Сбор и анализ требований
   3. Определение сроков
   4. Выбор технологий
   5. Анализ проектов схожей тематики
   6. Определение целей
2. Разработка приложения Подключение базы данных
   1. Работа над базами данных
   2. Создание прототипа интерфейса
   3. Реализация отчетов и работы с ними
   4. Front-end разработка
   5. Работа над интерфейсом
   6. Back-end разработка
   7. Оптимизация кода
3. Тестирование продукта
   1. Модульное тестирование
   2. Интеграционное тестирование
   3. Регрессионное тестирование
   4. Системное тестирование
4. Работа над обучением пользователей
   1. Подготовка документации для пользователей
   2. Создание FAQ
5. Работа на практиках
   1. Подготовка документации по проекту
   2. Выступления с докладом
6. Работы в команде по проекту
   1. Координация выполнения задач проекта
   2. Определение задач для каждого участника команды
   3. Встречи для вопросов по проекту
   4. Инициализация общего родительского репозитория и дочерних репозиториев

## Оценка трудозатрат на выполнение проекта

**Оценка трудоемкости по WBS:**

|  |  |  |
| --- | --- | --- |
| Наименование задачи | Наименование работ | Трудозатраты (человеко-часы) |
| Инициализация проекта | Формирование команды | 2 |
| Сбор и анализ требований | 10 |
| Определение сроков | 2 |
| Выбор технологий | 7 |
| Анализ проектов схожей тематики | 8 |
| Определение целей | 8 |
| Разработка приложения | Подключение базы данных | 6 |
| Работа над базами данных | 5 |
| Создание прототипа интерфейса | 45 |
| Реализация отчетов и работы с ними | 40 |
| Front-end разработка | 70 |
| Работа над интерфейсом | 18 |
| Back-end разработка | 160 |
| Оптимизация кода | 100 |
| Тестирование продукта | Модульное тестирование | 20 |
| Интеграционное тестирование | 8 |
| Регрессионное тестирование | 4 |
| Системное тестирование | 4 |
| Работа над обучением пользователей | Подготовка документации для пользователей | 8 |
| Создание FAQ | 3 |
| Работа на практиках | Подготовка документации по проекту | 18 |
| Выступления с докладом | 4 |
| Работы в команде по проекту | Координация выполнения задач проекта | 20 |
| Определение задач для каждого участника команды | 4 |
| Встречи для вопросов по проекту | 30 |
| Инициализация общего родительского репозитория и дочерних репозиториев | 4 |
| Итого |  | 788 |

**Оценка трудоемкости по UCP:**

**1. Определение весовых показателей действующих лиц.**

В системе 3 основных типа действующих лиц: менеджер, работник и горничная. Они общаются с системой через графический интерфейс пользователя, поэтому они относятся к сложному типу действующих лиц. Весовой коэффициент каждого из этих трех действующих лиц — 3.

Система — действующее лицо, взаимодействующее через программный интерфейс. То есть, действующее лицо простого типа с весовым коэффициентом 1.

**2. Вычисление неурегулированного весового показателя действующих лиц (Unadjusted Actors Weight, UAW).**

UAW = 3\*3+1=10

**3. Определение весовых показатели вариантов использования**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Варианты использования | Количество Use Case транзакций | Вес |
| 1 | Вход в систему | 4 | 10 |
| 2 | Регистрация нового пользователя | 5 | 10 |
| 3 | Просмотр списка услуг | 2 | 5 |
| 4 | Создание нового заказа | 3 | 5 |
| 5 | Просмотр невыполненных заказов | 2 | 5 |
| 6 | Изменение данных услуги | 3 | 5 |
| 7 | Назначение статуса пользователю | 4 | 10 |
| 8 | Выполнение заказа | 3 | 5 |
| 9 | Управление пользователями | 6 | 10 |
| 10 | Просмотр выполненных заказов | 2 | 5 |
|  | Итого |  | 70 |

**4. Определение неурегулированных весовых показателей вариантов использования —UUCW (Unadjusted Use Case Weight)**

В соответствии с таблицей UUCW = 70

**5. Определение неурегулированного общего показателя Total UUCP**

Total UUCP = Total UAW + Total UUCW = 10 + 70 =80

**6. Определение технической сложности проекта.** Рассчитаем показатель технической сложности TCF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель | Технический фактор | Вес | Значение | Вес \* Знач.  ( Весi\*Ti) |
| Т1 | Распределенность системы | 2 | 0 | 0 |
| Т2 | Высокая производительность (пропускная способность) | 1 | 1 | 1 |
| T3 | Ориентация на эффективность конечных пользователей | 1 | 1 | 1 |
| Т4 | Сложная обработка данных | 1 | 0 | 0 |
| Т5 | Повторное использование кода | 1 | 1 | 1 |
| Т6 | Простота установки | 0,5 | 1 | 0,5 |
| Т7 | Простота использования | 0,5 | 2 | 1 |
| Т8 | Переносимость | 2 | 1 | 2 |
| Т9 | Простота внесения изменений | 1 | 2 | 2 |
| T10 | Параллельная работа | 1 | 1 | 1 |
| T11 | Специальные требования к безопасности | 1 | 0 | 0 |
| T12 | Непосредственный доступ к системе со стороны внешних организаций | 1 | 1 | 1 |
| T13 | Специальные требования к обучению пользователей | 1 | 0 | 0 |
|  | Итого |  |  | 10.5 |

TCF=0,6+(0,01\*=0.6+0.01\*10.5=0.7

**7. Определение влияния условий разработки**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель | Фактор | Вес | Значение | Вес \* Знач.  ( Весi\*Fi) |
| F1 | Знакомство с проектом | 1,5 | 5 | 7.5 |
| F2 | Опыт разработки приложений | 0,5 | 2 | 1 |
| F3 | Опыт использования объектно-ориентированного подхода | 1 | 3 | 3 |
| F4 | Наличие ведущего аналитика | 0,5 | 2 | 1 |
| F5 | Мотивация | 1 | 5 | 5 |
| F6 | Стабильность требований | 2 | 3 | 6 |
| F7 | Частичная занятость | -1 | 0 | 0 |
| F8 | Сложные языки программирования | -1 | 0 | 0 |
| Итого |  |  |  | 23.5 |

EF=1,4+(-0,03\*=1,4+(-0,03\*23.5)=0,69

**8. Расчет UCP**. UCP = UUCP \* TCF \* EF = 80\*0,07\*0,69 =38.6

**9. Оценка трудоемкости проекта**

Так как количество показателей F1–F6, имеющих значение меньше 3, равно 2 и показателей F7–F8, имеющих значение больше 3, равно 0, примем производительность равной 20 чел/час на одну UCP. Тогда трудоемкость проекта составят 772часов.

## Перечень работ по проекту, выполненных автором курсовой работы

**1. Работа с базами данных**

В рамках проекта было проведено полное взаимодействие с базами данных, включающее следующие этапы:

Проектирование структуры базы данных: Разработка схемы данных, создание таблиц и определение связей между ними для обеспечения корректного хранения информации.

Оптимизация запросов: Модификация SQL-запросов для повышения их эффективности, минимизация времени отклика.

Интеграция с существующими системами: Обеспечение корректной работы базы данных с другими компонентами системы, настройка подключения и синхронизации данных.

Резервное копирование и восстановление данных: Настройка процедур бэкапов для обеспечения сохранности данных в случае непредвиденных ситуаций.

Реализация механизма каскадного удаления для автоматического удаления связанных записей в других таблицах при удалении записи в основной таблице. Это обеспечило согласованность данных и исключило "осиротевшие" записи, которые могли бы нарушить целостность базы данных.

Разработка и настройка внешних ключей для обеспечения связности данных между таблицами. Взаимодействие между таблицами реализовано через правильное определение и настройку связей "один ко многим", "многие ко многим" и "один к одному", что обеспечило целостность данных и позволило избежать ошибок при вставке, обновлении и удалении записей.

**2. Создание прототипа интерфейса**

Создание прототипа интерфейса с использованием JavaFX. Этот прототип позволил визуализировать ключевые функциональные блоки и элементы управления, которые должны быть реализованы в финальной версии приложения.

Прототип интерфейса использовался для проведения первичных тестирований удобства использования и взаимодействия с системой.

**3. Оптимизация кода**

В рамках оптимизации кода были выполнены следующие работы:

Частично проведен анализ существующего кода для выявления узких мест и слабых мест в производительности.

Оптимизация структуры кода, устранение дублирования, улучшение читаемости и поддерживаемости.

Модификация существующих алгоритмов для повышения их эффективности. Внедрение более быстрых и экономичных решений для обработки данных.

**4. Бэкенд имплементация**

В рамках разработки бэкенд-логики системы были выполнены следующие ключевые этапы и задачи:

1. Проектирование и реализация основного компонента приложения (Main)

В рамках бэкенд-имплементации был реализован главный компонент запуска системы — класс Main. Это точка входа в приложение, которая запускает процесс работы с графическим интерфейсом и серверной логикой. Этот компонент инициализирует создание объекта Application, который управляет всеми аспектами взаимодействия с пользователем через графический интерфейс.

1. Обработка взаимодействия с пользователем через интерфейс (Application)

Важной частью бэкенд-имплементации является компонент Application, который отвечает за управление графическим интерфейсом и взаимодействие с пользователями через различные панели. Помимо этого все SQL запросы генерируются в нем.

1. Обеспечение безопасности через хэширование паролей (HashUtil)

Одним из важных аспектов бэкенд-имплементации является безопасность хранения и проверки паролей пользователей. Для этого был создан компонент HashUtil, который использует библиотеку BCrypt для хэширования паролей.

Этот подход обеспечивает надежную защиту пользовательских данных и предотвращает хранение паролей в открытом виде, что критически важно для безопасности системы.

1. Работа с конфигурацией и настройками системы (Config)

Компонент Config реализует обработку параметров конфигурации, таких как параметры подключения к базе данных и другие важные настройки системы. Этот компонент играет ключевую роль в обеспечении гибкости и удобства конфигурации без необходимости изменения кода.

**5. Проведение тестирования**

Проведение функционального, регрессионного тестирования. Исправление ошибок, если они были найдены

# Проектирование структуры и компонентов программного продукта

## Архитектура разрабатываемого ПО

Основные компоненты системы:

* **Графический интерфейс пользователя (GUI)**: Отвечает за взаимодействие с пользователем. Пользователь может вводить данные, инициировать действия (например, создать заказ, просматривать список услуг, специалистов и заказов) и получать информацию о результатах своих действий.
* **Компонент обработки данных (Backend)**: Этот компонент выполняет все операции с базой данных, такие как извлечение, обновление и добавление данных. Он обрабатывает запросы от GUI и возвращает результаты.
* **База данных**: Хранит все данные системы, такие как информация об услугах, специалистах, заказах и квартирах.

Связи между компонентами:

* **GUI и Backend**: Взаимодействие между ними происходит через события, такие как нажатие кнопок, заполнение форм и запросы к базе данных. GUI отправляет запросы на обработку данных в Backend, который затем взаимодействует с базой данных и возвращает результаты.
* **Backend и База данных**: База данных используется для хранения и извлечения данных, с помощью SQL-запросов. Backend выполняет запросы, получая данные из таблиц и записывая их обратно.

Изображение выглядит как текст, диаграмма, зарисовка, План

Автоматически созданное описание

**2. Краткое описание компонентов**

* **GUI (Графический интерфейс пользователя)**:
  + Содержит панели для отображения списка услуг, специалистов и заказов.
  + Предоставляет формы для ввода данных при заказе новой услуги.
  + Отправляет запросы на выполнение операций через кнопки и обработчики событий.
* **Backend (Обработка данных)**:
  + Обрабатывает логику работы системы, включая валидацию данных, выполнение SQL-запросов и управление состоянием приложения.
  + Осуществляет взаимодействие с базой данных: извлечение данных (список услуг, специалистов, заказов) и выполнение изменений (создание заказов, добавление адресов).
* **База данных**:
  + Хранит все данные, включая таблицы services, specialists, service\_requests, completed\_orders, и apartments.
  + Структура базы данных поддерживает взаимосвязи между таблицами, что позволяет эффективно извлекать и обрабатывать данные.

**В UML диаграммах GUI представлен как USER**

Регистрация аккаунта

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Вход в аккаунт  
Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Смена специалиста в заявке  
Изображение выглядит как текст, снимок экрана, Параллельный, число

Автоматически созданное описание

Выполнение заказа  
Изображение выглядит как текст, чек, Параллельный, снимок экрана

Автоматически созданное описание

Изменение данных пользователя  
Изображение выглядит как текст, чек, диаграмма, Параллельный

Автоматически созданное описание

Изменение данных администратора

Изображение выглядит как текст, диаграмма, чек, Параллельный

Автоматически созданное описание

Изменение данных специалистов  
Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Изменение данных услуг

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Автоматически созданное описание

Редактирование данных заказов  
Изображение выглядит как текст, чек, Параллельный, диаграмма

Автоматически созданное описание

Главное меню  
Изображение выглядит как текст, чек, Параллельный, документ

Автоматически созданное описание

## Модель предметной области

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

## Описание интерфейса пользователя

Интерфейс пользователя состоит из ряда окон, созданных с использованием библиотеки Swing. Он предоставляет интуитивно понятный доступ к функционалу приложения с учетом роли пользователя (администратор, специалист или обычный пользователь).

**Главное окно (Авторизация)**

* **Назначение:** Позволяет пользователю войти в систему или зарегистрировать новую учетную запись.
* **Элементы интерфейса:**
  + **Заголовок:** Отображает приветственное сообщение *"WELCOME!"*.
  + **Кнопки:**
    - LOG IN: Для входа в существующую учетную запись.
    - SIGN UP: Для создания новой учетной записи.
  + **Фон:** Использует градиенты и эффекты шумов для визуальной привлекательности.

**Панель администратора**

* **Назначение:** Предоставляет администратору инструменты управления данными приложения.
* **Элементы интерфейса:**
  + **Список кнопок:**
    - View the list of orders: Просмотр всех заказов.
    - Make an action with list of orders: Управление заказами.
    - View the list of services: Просмотр всех услуг.
    - Make an action with list of services: Управление услугами.
    - View the list of users: Просмотр обычных пользователей.
    - Make an action with list of users: Управление пользователями.
    - View the list of workers: Просмотр специалистов.
    - Delete workers from db: Удаление специалистов.
    - View the list of administrators: Просмотр администраторов.
    - Delete administrators from db: Удаление администраторов.
    - Unfulfilled orders: Просмотр невыполненных заказов.
    - Completed orders: Просмотр выполненных заказов.
  + **Фон:** Градиент с плавными переходами и мелкими элементами декора.

**Панель специалиста**

* **Назначение:** Предоставляет доступ к заказам, связанным с компетенцией специалиста.
* **Элементы интерфейса:**
  + **Список кнопок:**
    - View your outstanding orders: Просмотр заказов специалиста.
    - Send the completed order id: Отправка завершенного заказа.
    - View the orders you can complete: Просмотр доступных заказов.
    - Make an action with list of services: Управление назначенными услугами.
  + **Фон:** Градиент с динамическим распределением точек и оттенков.

**Панель пользователя**

* **Назначение:** Позволяет пользователю взаимодействовать с приложением в пределах предоставленных ему полномочий.
* **Элементы интерфейса:**
  + Окно с функциями, адаптированными под роль обычного пользователя.
  + Визуальные эффекты**:** Градиенты, соответствующие стилю приложения.

**Дополнительные окна**

* **Окно регистрации:**
  + Поля ввода для логина и пароля.
  + Проверка уникальности логина и сохранение данных пользователя в базе.
* **Окно входа:**
  + Поля ввода логина и пароля.
  + Проверка учетных данных с использованием хэширования пароля.
  + Информативные сообщения об ошибках.

**Особенности интерфейса**

1. **Адаптивный дизайн:**
   * Градиенты и шумовые эффекты для улучшения восприятия.
   * Большие и заметные кнопки с уникальной стилизацией.
2. **Разделение ролей:**
   * Каждый пользователь видит интерфейс, соответствующий его полномочиям.
3. **Фокус на удобстве:**
   * Простота взаимодействия через кнопки и всплывающие окна.
   * Все действия подтверждаются уведомлениями (успешность, ошибки).

## Описание схемы БД

База данных, используемая в проекте, включает следующие таблицы:

1) Данные об апартаментах (apartments)

Эта таблица содержит информацию об объектах, куда вызываются специалисты. В ней хранятся:

Уникальный идентификатор апартаментов (id);

Адрес апартаментов (address).

2) Данные о выполненных заявках (completed\_orders)

Таблица содержит информацию о завершённых заказах. Поля таблицы:

Уникальный идентификатор заказа (id);

Дата создания заказа (order\_creation\_date);

Дата завершения заказа (order\_completion\_date);

Комментарий клиента о выполнении услуги (user\_comment);

Идентификатор услуги, оказанной в рамках заказа (service\_id);

Идентификатор специалиста, выполнившего заказ (specialist\_id).

3) Данные об услугах (services)

Таблица хранит список услуг, предоставляемых клиентам. В ней содержатся:

Уникальный идентификатор услуги (id);

Название услуги (name).

4) Данные о специалистах (specialists)

В таблице хранится информация о специалистах, предоставляющих услуги. Поля таблицы:

Уникальный идентификатор специалиста (id);

Полное имя специалиста (full\_name);

Список идентификаторов услуг, которые может выполнять специалист (service\_id);

Логин специалиста для входа в систему (login).

5) Заявки на обслуживание (service\_requests)

Таблица фиксирует заявки клиентов на предоставление услуг. Содержит следующие данные:

Уникальный идентификатор заявки (id);

Идентификатор услуги, на которую оформлена заявка (service\_id);

Идентификатор специалиста, назначенного на заявку (specialist\_id);

Дата подачи заявки (request\_date);

Идентификатор апартаментов, куда специалист должен приехать (apartment\_id).

6) Данные о пользователях (users)

Эта таблица содержит учетные записи пользователей системы. В ней хранятся:

Уникальный идентификатор пользователя (id);

Логин для входа в систему (login);

Пароль пользователя в зашифрованном виде (password);

Ниже приведен рисунок, на котором видны все таблицы.

Признак, указывающий, является ли пользователь администратором (is\_admin).Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Автоматически созданное описание

## Проектирование компонентов

**1. Основные компоненты системы**

**1.1 Главный компонент запуска (Main)**

Класс: Main  
Назначение:

* Это точка входа в приложение. Здесь создаётся экземпляр класса Application, который управляет основным окном программы. Также здесь запускается поток Swing, необходимый для корректной работы графического интерфейса.

Методы:

* main(String[] args) — Метод запускает приложение в потоке Swing, создавая объект Application и устанавливая его видимость.

Взаимодействие с другими компонентами:

* Этот класс взаимодействует только с классом Application, который инициализирует GUI и запускает основные операции приложения.

**1.2 Графический пользовательский интерфейс (GUI)**

Класс: Application  
Назначение:

* Отображает основной интерфейс приложения и управляет окнами для входа, регистрации пользователей, а также для администраторов, специалистов и обычных пользователей. Взаимодействует с пользователем через интерфейс Swing.
* Обрабатывает аутентификацию для пользователей с различными ролями (администратор, специалист, обычный пользователь).
* Использует CardLayout для динамического переключения между панелями, такими как окно входа, окно регистрации, а также панели для различных ролей.

Основные методы:

* createLoginPanel() — Создаёт и настраивает панель для входа пользователя в систему.
* createAccount() — Создаёт и настраивает панель для регистрации нового пользователя.
* createLoginButton() — Метод для создания кнопок с индивидуальным дизайном (для входа и регистрации).
* logToAccount() — Обрабатывает логин пользователя, проверяя его в базе данных.
* showUserPanel(), showAdminPanel(), showSpecialistPanel() — Показывает соответствующие панели для разных ролей пользователей.

Взаимодействие с другими компонентами:

* Application взаимодействует с компонентами для работы с данными пользователей и базы данных через JDBC.
* Взаимодействует с классом HashUtil для проверки и создания хешей паролей.
* Использует класс Config для настройки подключения к базе данных.

**1.3 Модуль хэширования паролей (HashUtil)**

Класс: HashUtil  
Назначение:

* Обеспечивает безопасность паролей, хэшируя их с использованием библиотеки BCrypt. Это гарантирует защиту паролей пользователей при их хранении и проверке.

Методы:

* hash(String getIt) — Генерирует хэш для переданного пароля.
* check(String getIt, String hashedInfo) — Проверяет введённый пароль на соответствие сохранённому хэшу.

Взаимодействие с другими компонентами:

* Взаимодействует с компонентами, где необходимо хранить или проверять пароли пользователей (например, при регистрации и входе).

**1.4 Работа с конфигурацией (Config)**

Класс: Config  
Назначение:

* Загружает параметры конфигурации из файла config.properties, такие как настройки для подключения к базе данных (например, URL, логин и пароль). Этот компонент облегчает конфигурацию приложения без необходимости изменения кода.

Методы:

* loadProperties() — Загружает конфигурацию из файла и возвращает объект Properties с параметрами.

Взаимодействие с другими компонентами:

* Взаимодействует с классом Application, передавая данные для подключения к базе данных.

**1.5 Работа с базой данных (Database)**

Класс: Application  
Назначение:

* Обеспечивает взаимодействие с базой данных, выполняя операции с данными пользователей, услуг, заказов и специалистов. Этот компонент управляет подключением к базе данных и выполняет SQL-запросы для получения, обновления и удаления данных.
* Через JDBC соединение, класс Application обрабатывает запросы, связанные с регистрацией пользователей, аутентификацией, а также управлением заказами и услугами.

Методы:

* createAccount() — Метод регистрации нового пользователя.
* logToAccount() — Метод для аутентификации пользователя.
* checkLoginExists() — Проверяет, существует ли уже логин в базе данных.
* modifyOrdersData(), modifyUsersData() и другие — Методы для изменения различных данных в базе.

Взаимодействие с другими компонентами:

* Взаимодействует с классом Config, который предоставляет параметры подключения к базе данных.
* Использует JDBC для выполнения SQL-запросов через PreparedStatement и обработки результатов с помощью ResultSet.
* Использует данные из базы для отображения информации в интерфейсе (например, список пользователей, заказов, услуг и специалистов).

**2. Взаимодействие между компонентами**

1. Main → Application:
   * Главный класс вызывает Application для отображения окна.
2. Application ↔ HashUtil:
   * Application использует методы хэширования для защиты паролей при регистрации и входе.
3. Application ↔ База данных (через JDBC):
   * Все операции с базой данных (например, регистрация пользователя, вход в систему) выполняются через JDBC, используя данные, полученные через Config.
   * Создаёт соединение с базой данных, выполняет запросы через PreparedStatement и обрабатывает результаты через ResultSet.
4. Config → Файл конфигурации:
   * Считывает настройки подключения к базе данных из файла config.properties.

**3. Архитектура приложения**

Модульность

GUI**-слой (Представление):**

* Вся работа с пользовательским интерфейсом (UI), включая создание панелей, кнопок, обработку визуальных элементов и взаимодействие с пользователем, реализована в классе Application.
* Этот слой отвечает исключительно за отображение данных и взаимодействие с пользователем.
* Использование CardLayout для управления сменой панелей позволяет создать удобную навигацию, разделяя разные экраны для входа, регистрации и работы с ролями пользователей.

**Бизнес-логика:**

* Логика обработки данных и принятия решений, связанная с пользователями, заказами и специалистами, находится в методах класса Application.
* В этом слое реализуются все ключевые операции, такие как регистрация и аутентификация пользователей, создание и изменение заказов, управление услугами.

**Доступ к данным (Модуль доступа к базе данных):**

* В текущей реализации доступ к данным осуществляется непосредственно через методы класса Application, который использует JDBC для взаимодействия с базой данных.
* Параметры подключения к базе данных загружаются из конфигурационного файла config.properties.

**Конфигурация (Управление настройками и параметрами):**

* Все параметры, касающиеся конфигурации приложения, такие как параметры подключения к базе данных, хранятся в конфигурационном файле (например, config.properties).
* Этот подход позволяет:
  + Легко изменять настройки (например, изменить строку подключения к базе данных) без необходимости изменения исходного кода.
  + Повышает гибкость системы, позволяя адаптировать её к различным средам (например, тестовая или производственная база данных) через изменение конфигурационного файла.
  + Централизует управление параметрами, что упрощает их настройку и управление ими в будущем.

## Описание процесса кодирования

Последовательность действий (этапы) при кодировании с использованием ИИ

1. Перед началом разработки был определён набор задач:

1. Создать функционал для регистрации нового пользователя через графический интерфейс.
2. Обеспечить базовую валидацию введённых данных (например, проверку пустых полей).
3. Реализовать проверку на существование логина в базе данных.
4. Добавить хэширование паролей для безопасности.
5. Создание шаблонов кода и прототипирование с помощью ИИ

2. Создание шаблонов кода и прототипирование с помощью ИИ

Для начала был использован искусственный интеллект, который сгенерировал начальный прототип функции создания аккаунта. Вот пример, как нейросеть изначально предложила код:

private void createAccount() {

JFrame logFrame = new JFrame("Create Account");

logFrame.setSize(400, 200);

logFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

logFrame.setLocationRelativeTo(this);

JPanel inputPanel = new JPanel(new GridLayout(3, 2, 10, 10));

JTextField loginField = new JTextField();

JPasswordField passwordField = new JPasswordField();

inputPanel.add(new JLabel("Login:"));

inputPanel.add(loginField);

inputPanel.add(new JLabel("Password:"));

inputPanel.add(passwordField);

JButton addButton = new JButton("Create");

addButton.addActionListener(e -> {

String login = loginField.getText();

String password = new String(passwordField.getPassword());

// Basic validation

if (login.trim().isEmpty() || password.trim().isEmpty()) {

JOptionPane.showMessageDialog(null, "Fill in all fields");

return;

}

// Placeholder for adding user to the database

System.out.println("User created: " + login);

});

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

buttonPanel.add(addButton);

logFrame.add(inputPanel, BorderLayout.NORTH);

logFrame.add(buttonPanel, BorderLayout.SOUTH);

logFrame.setVisible(true);

}

Изначальный код был простым, демонстрировал основные элементы интерфейса и базовую валидацию, но не включал логику взаимодействия с базой данных и защиту данных.

3. Рефакторинг и оптимизация кода

Были внесены изменения, чтобы сделать код более функциональным и безопасным:

1. Добавил подключение к базе данных для проверки существующего логина и сохранения данных нового пользователя.
2. Добавил вызов функции хэширования пароля через утилиту HashUtil.hash.
3. Реализовал обработку ошибок при работе с базой данных.

Итоговый код после доработок выглядит так:

private void createAccount() {

JFrame logFrame = new JFrame("Create Account");

logFrame.setSize(400, 200);

logFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

logFrame.setLocationRelativeTo(this);

JPanel inputPanel = new JPanel(new GridLayout(3, 2, 10, 10));

JTextField loginField = new JTextField();

JPasswordField passwordField = new JPasswordField();

inputPanel.add(new JLabel("Login:"));

inputPanel.add(loginField);

inputPanel.add(new JLabel("Password:"));

inputPanel.add(passwordField);

JButton addButton = new JButton("Create");

addButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String login = loginField.getText();

String password = new String(passwordField.getPassword());

if (login.trim().isEmpty() || password.trim().isEmpty()) {

JOptionPane.showMessageDialog(null, "Fill in all the fields");

return;

}

try {

if (checkLoginExists(login)) {

JOptionPane.showMessageDialog(null, "User with this login already exists!");

return;

}

PreparedStatement statement = connection.prepareStatement(

"INSERT INTO users (login, password) VALUES(?,?)"

);

statement.setString(1, login);

statement.setString(2, HashUtil.hash(password));

int rowsAffected = statement.executeUpdate();

if (rowsAffected > 0) {

JOptionPane.showMessageDialog(null, "The account has been successfully created!");

logFrame.dispose();

} else {

JOptionPane.showMessageDialog(null, "Error occurred when creating an account", "Error", JOptionPane.ERROR\_MESSAGE);

}

} catch (SQLException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Error occurred when creating an account: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

buttonPanel.add(addButton);

logFrame.add(inputPanel, BorderLayout.NORTH);

logFrame.add(buttonPanel, BorderLayout.SOUTH);

logFrame.setVisible(true);

}

4. Генерация тестов и проверка

ИИ также помог предложить тестовые сценарии для проверки функциональности:

Тесты на валидацию логина и пароля:

* Попытка зарегистрировать пользователя с пустым логином и паролем.
* Ввод логина с недопустимыми символами, такими как @, #, $, % (если запрещено).
* Ввод пароля, содержащего только пробелы (например, " ").
* Проверка минимальной длины пароля (например, попытка использовать пароль длиной менее 8 символов).
* Проверка максимальной длины пароля (например, пароль длиной 256 символов).
* Попытка использовать логин, состоящий только из цифр.
* Регистрация пользователя с паролем, содержащим символы других языков (например, кириллица или китайские иероглифы).

Тесты на дубликаты:

* Попытка зарегистрировать двух пользователей с одинаковыми логинами.
* Регистрация логина, который отличается только регистром букв (например, admin и Admin).

Тесты на пользовательский интерфейс (UI):

* Убедиться, что после успешной регистрации окно регистрации закрывается.
* Проверить, что кнопка "Create" становится доступной только после ввода всех обязательных полей.
* Попытка отправить форму с использованием клавиши "Enter", чтобы убедиться, что это тоже приводит к обработке регистрации.
* Проверка отображения сообщения об успехе после регистрации.

Человек разработал автоматические тесты на основе предложений ИИ для проверки работы функций через подключение к тестовой базе данных.

5. Поддержка и документирование кода

Были добавлены комментарии к коду, описывающие ключевые шаги и логику. Например:

* Объяснение использования функции хэширования.
* Пояснение, что проверка на существование логина производится перед добавлением нового пользователя.

Документация включает:

1. Список требований для базы данных (структура таблиц).
2. Описание, как обновлять хэш-функцию при изменении требований безопасности.
3. Примеры использования функции createAccount для других частей проекта.

# Выбор стратегии тестирования и разработка тестов

## Стратегия тестирования

**Подход к тестированию**

Для тестирования будут использованы следующие методы:

* **Мануальное тестирование**: Проверка требований из спецификации SRS.Подбор тестов вручную для проверки сложных и критичных сценариев.

**База тестирования**

* **Основой тестирования** выступают пользовательские требования покрывающие:
  + Работу с заказами для всех типов пользователей (администратор, работник, клиент).
  + Корректность отображения данных из таблиц БД.
  + Функциональность редактирования, удаления и обновления.

**Уровни тестирования**

1. **Модульное тестирование**: Проверка отдельных компонентов, например, запросов к базе данных и бизнес-логики.
2. **Интеграционное тестирование**: Тестирование взаимодействия между модулями (например, взаимодействие frontend-а с backend-ом).
3. **Системное тестирование**: Проверка всего приложения в сборке.
4. **Приемочное тестирование**: Убедиться, что приложение соответствует требованиям клиентов.

**Типы тестирования**

* **Функциональное тестирование**: Проверка реализованного функционала.
* **Регрессионное тестирование**: Убедиться, что новые изменения не сломали старую функциональность.
* **Тестирование безопасности**: Проверка ролей пользователей (админ, клиент, работник).

**Требования к окружению**

* **Операционная система**: Windows, поддерживающая psql (версия 15 и новее) и Java (21 и новее).
* **База данных**: PostgreSQL 15 и новее.
* **Среда исполнения**: Java 21 и новее.

**Инструменты тестирования**

* **TMS**: TestRail для управления тестовыми сценариями.

**Результаты тестирования**

Документация:

* **Перед тестами**: Тест-план, список тестовых сценариев.
* **Во время тестов**: Тестовые прогоны, отчеты о багах.
* **После тестов**: Сводный отчет о тестировании, включая найденные баги и метрики покрытия.

**Отчетность**

* **Виды отчетов**:
  + Ежедневные отчеты о ходе тестирования.
  + Еженедельные сводки по прогрессу, включая список критичных и исправленных багов.

**Результаты тестов**

* Сводные отчеты генерируются:
  + **Еженедельно**: Подробный отчет о ходе тестирования, метриках покрытия и основных проблемах.

## Описание процесса модульного и интеграционного тестирования

**Описание процесса модульного и интеграционного тестирования**

**1. Последовательность действий при модульном тестировании**

1. **Выбор модуля**: Определяются отдельные функции или методы для проверки.
2. **Создание тест-кейсов**: Разрабатываются сценарии для проверки функционала каждого модуля, включая стандартные и граничные условия.
3. **Выполнение тестов**: Выполняются тест-кейсы вручную, проверяется соответствие фактических и ожидаемых результатов.
4. **Анализ результатов**: Ошибки фиксируются, исправляются, затем тесты выполняются повторно.

**2. Последовательность действий при интеграционном тестировании**

1. **Выбор компонентов для интеграции**: Определяются модули, которые необходимо протестировать в связке.
2. **Разработка интеграционных сценариев**: Создаются тест-кейсы для проверки передачи данных и взаимодействия между компонентами.
3. **Выполнение тестов**: Ручное выполнение тестов для проверки корректности взаимодействия модулей.
4. **Анализ и исправление ошибок**: Выявленные ошибки фиксируются и отправляются на доработку.

Все тесты проводятся вручную для ускорения процесса в связи с небольшим объемом работы

**Ручное тестирование на практике: процесс и примеры для метода orderNewService()**

Ручное тестирование — это процесс, при котором тестировщик вручную выполняет функциональные проверки приложения, вводя данные и наблюдая за результатами. Вот как это может выглядеть на практике:

**Подготовка к тестированию**

1. **Тестовая среда:**
   * Установлено тестовое окружение приложения.
   * Используется тестовая база данных с таблицами apartments, specialists, и service\_requests.
   * Заданы тестовые данные:
     + Пример адреса: Main Street, 10, 5.
     + Пример специалиста: ID = 1, поддерживает услугу ID = 2.
2. **Тестовый сценарий:**
   * Составлены сценарии, которые покрывают все функциональные возможности метода, включая как положительные, так и отрицательные случаи.

**Шаги ручного тестирования**

**1. Проверка валидации полей ввода:**

* **Шаги:**
  1. Открыть форму заказа.
  2. Оставить все поля пустыми и нажать кнопку "Make an order".
* **Ожидаемый результат:**
  1. Отображается сообщение: *"Fill at least in service, street, and house fields"*.
  2. Заказ не создаётся.
* **Шаги:**
  1. Заполнить поле Service ID текстом (например, abc) и нажать "Make an order".
* **Ожидаемый результат:**
  1. Отображается сообщение: *"The Service ID and Specialist ID fields must contain numeric values"*.

**2. Создание заказа с существующим адресом:**

* **Шаги:**
  1. Заполнить поля:
     + Service ID: 2.
     + Street: Main Street.
     + House: 10.
     + Apartment number: 5.
  2. Нажать "Make an order".
* **Ожидаемый результат:**
  1. Заказ успешно создаётся.
  2. Отображается сообщение: *"The order has been successfully placed"*.

**3. Создание заказа с новым адресом:**

* **Шаги:**
  1. Заполнить поля:
     + Service ID: 2.
     + Street: Second Street.
     + House: 15.
     + Apartment number: 7.
  2. Нажать "Make an order".
  3. В появившемся диалоге подтвердить добавление нового адреса (нажать "Yes").
* **Ожидаемый результат:**
  1. Новый адрес добавляется в базу данных.
  2. Заказ успешно создаётся.

**4. Отмена добавления нового адреса:**

* **Шаги:**
  1. Заполнить поля с новым адресом, как в предыдущем случае.
  2. В диалоге подтверждения нового адреса выбрать "Нет".
* **Ожидаемый результат:**
  1. Новый адрес не добавляется.
  2. Отображается сообщение: *"Order canceled"*.
  3. Заказ не создаётся.

**5. Создание заказа с указанным специалистом:**

* **Шаги:**
  1. Заполнить поля:
     + Service ID: 2.
     + Specialist ID: 1.
     + Street: Main Street.
     + House: 10.
     + Apartment number: 5.
  2. Нажать "Make an order".
* **Ожидаемый результат:**
  1. Проверяется, что специалист поддерживает услугу.
  2. Заказ создаётся с указанным специалистом.

**6. Ошибка при несоответствии специалиста и услуги:**

* **Шаги:**
  1. Заполнить поля:
     + Service ID: 3 (услуга, не поддерживаемая специалистом).
     + Specialist ID: 1.
     + Остальные данные как в предыдущем случае.
  2. Нажать "Make an order".
* **Ожидаемый результат:**
  1. Отображается сообщение: *"The selected specialist cannot perform the specified service."*
  2. Заказ не создаётся.

**7. Ошибка базы данных:**

* **Шаги:**
  1. Отключить базу данных или симулировать её недоступность.
  2. Попробовать создать заказ с корректными данными.
* **Ожидаемый результат:**
  1. Отображается сообщение: *"Database connection error"*.
  2. Приложение корректно откатывает транзакцию.

**Документирование тестирования**

Для каждого тестового сценария фиксируются результаты:

* Успешные тесты отмечаются как "Пройдено".
* Ошибки документируются с указанием:
  + Данных ввода.
  + Ожидаемого и фактического результата.
  + Скриншотов или лога ошибок.

**Пример отчёта о тестировании**

| **Тестовый сценарий** | **Данные ввода** | **Ожидаемый результат** | **Фактический результат** | **Статус** |
| --- | --- | --- | --- | --- |
| Проверка пустых полей | Все поля пустые | Сообщение о заполнении обязательных полей | Совпадает | Пройдено |
| Ввод некорректного Service ID | Service ID = abc | Сообщение о неверном формате | Совпадает | Пройдено |
| Создание заказа с существующим адресом | Service ID = 2, Street = Main Street | Заказ создаётся | Совпадает | Пройдено |
| Создание заказа с новым адресом | Service ID = 2, Street = New Street | Новый адрес добавляется, заказ создаётся | Совпадает | Пройдено |
| Отмена добавления нового адреса | Service ID = 2, Street = Cancel Street | Заказ не создаётся | Совпадает | Пройдено |
| Ошибка специалиста и услуги | Service ID = 3, Specialist ID = 1 | Сообщение о несоответствии | Совпадает | Пройден |

## Тест-кейсы

**Задание 10. Тест-кейсы**

Тест-кейсы с пользовательскими требованиями

**Тест-кейс 1: Проверка успешного выполнения заказа администратором**

|  |  |
| --- | --- |
| **Приоритет** | A |
| **Связанное требование** | R01 (Администратор может обновлять статус заказа) |
| **Модуль и подмодуль** | Модуль: Управление заказами. Подмодуль: Изменение статуса. |
| **Заглавие тест-кейса** | Проверка успешного изменения статуса заказа на "Выполнено". |
| **Исходные данные** | В системе есть заказ со статусом "Не выполнено". Логин и пароль администратора. |
| **Шаги тест-кейса** | 1. Авторизоваться в системе под учетной записью администратора.  Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение  Автоматически созданное описание 2. Перейти в раздел "Управление заказами".  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание 3. Выбрать заказ со статусом "Не выполнено".  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание 4. Нажать "Изменить статус".  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание 5. Установить статус "Выполнено" и нажать "Сохранить".  Изображение выглядит как текст, снимок экрана, Шрифт, число  Автоматически созданное описание |
| **Ожидаемый результат** | 1. Статус заказа изменяется на "Выполнено".  2. Заказ исчезает из списка "Не выполнено".  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание |

**Тест-кейс 2: Проверка просмотра списка выполненных заказов клиентом**

|  |  |
| --- | --- |
| **Приоритет** | B |
| **Связанное требование** | R02 (Клиент может просматривать выполненные заказы). |
| **Модуль и подмодуль** | Модуль: Просмотр заказов. Подмодуль: Выполненные заказы. |
| **Заглавие тест-кейса** | Проверка отображения выполненных заказов в интерфейсе клиента. |
| **Исходные данные** | В системе есть выполненные заказы, связанные с клиентом. Логин и пароль клиента. |
| **Шаги тест-кейса** | 1. Авторизоваться в системе под учетной записью клиента. Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт     Автоматически созданное описание 2. Найти вкладку "Выполненные заказы".   Изображение выглядит как текст, снимок экрана, дизайн  Автоматически созданное описание  3. Выбрать вкладку "Выполненные заказы".  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание |
| **Ожидаемый результат** | Отображается список выполненных заказов клиента с корректными данными (даты начала и завершения, услуги, специалиста).  Изображение выглядит как текст, снимок экрана, число, Шрифт  Автоматически созданное описание |
|  |

**Тест-кейс 3: Проверка возможности специалиста изменить статус заказа на "Выполнено"**

|  |  |
| --- | --- |
| **Приоритет** | A |
| **Связанное требование** | R03 (Специалист может завершать заказы). |
| **Модуль и подмодуль** | Модуль: Управление заказами. Подмодуль: Завершение заказа. |
| **Заглавие тест-кейса** | Проверка успешного завершения заказа специалистом. |
| **Исходные данные** | В системе есть заказ, назначенный специалисту, со статусом "Не выполнено". Логин специалиста. |
| **Шаги тест-кейса** | 1. Авторизоваться в системе под учетной записью специалиста.  Изображение выглядит как текст, снимок экрана, Бренд, Шрифт  Автоматически созданное описание 2. Перейти в раздел "Мои заказы".  Изображение выглядит как текст, снимок экрана, линия, Шрифт  Автоматически созданное описание 3. Нажать "Изменить статус".  Изображение выглядит как текст, Шрифт, снимок экрана, линия  Автоматически созданное описание 4. Установить статус "Выполнено" и подтвердить. |
| **Ожидаемый результат** | Статус заказа изменяется на "Выполнено", заказ исчезает из списка "Не выполнено".  Изображение выглядит как текст, снимок экрана, Шрифт, число  Автоматически созданное описание |

**Тест-кейс 4: Проверка удаления услуги администратором**

|  |  |
| --- | --- |
| **Приоритет** | A |
|  |  |
| **Связанное требование** | R04 (Администратор может удалять услуги). |
| **Модуль и подмодуль** | Модуль: Управление услугами. Подмодуль: Удаление услуги. |
| **Заглавие тест-кейса** | Проверка удаления услуги из списка доступных. |
| **Исходные данные** | В системе есть услуга с уникальным идентификатором. Логин и пароль администратора. |
| **Шаги тест-кейса** | 1. Авторизоваться в системе под учетной записью администратора.  Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение  Автоматически созданное описание 2. Перейти в раздел "Услуги".  Изображение выглядит как текст, Шрифт, число, линия  Автоматически созданное описание 3. Найти услугу, которую нужно удалить. 4. Нажать на кнопку "Удалить".  Изображение выглядит как текст, снимок экрана, Шрифт, число  Автоматически созданное описание 5. Подтвердить удаление услуги.  Изображение выглядит как текст, снимок экрана, Шрифт, число  Автоматически созданное описание |
| **Ожидаемый результат** | Услуга удаляется из списка, и она больше не отображается в интерфейсе пользователей.  Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система  Автоматически созданное описание |

**Тест-кейс 5: Проверка создания запроса на выполнение услуги клиентом**

|  |  |
| --- | --- |
| **Приоритет** | A |
| **Связанное требование** | R05 (Клиент может создавать запросы на выполнение услуг). |
| **Модуль и подмодуль** | Модуль: Управление заказами. Подмодуль: Создание запроса. |
| **Заглавие тест-кейса** | Проверка создания нового запроса на выполнение услуги клиентом. |
| **Исходные данные** | В системе зарегистрирован клиент. Есть услуга, доступная для заказа. |
| **Шаги тест-кейса** | 1. Авторизоваться в системе под учетной записью клиента.  Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт  Автоматически созданное описание 2. Перейти в раздел "Сделать заказ".  Изображение выглядит как текст, снимок экрана, дизайн  Автоматически созданное описание 3. Выбрать услугу и указать данные.  4. Нажать кнопку "Заказать". 5. Подтвердить создание запроса.  Изображение выглядит как текст, снимок экрана, Шрифт, линия  Автоматически созданное описание |
| **Ожидаемый результат** | В системе создаётся новый запрос на выполнение услуги, который отображается в списке запросов клиента.Изображение выглядит как текст, снимок экрана, Шрифт, линия  Автоматически созданное описание |

# Заключение

В процессе работы над проектом мы стремились предоставить пользователям простой и эффективный инструмент для взаимодействия с профессионалами, необходимыми для выполнения различных бытовых задач, таких как ремонт, установка техники, обслуживание и другие услуги.

Одной из главных задач было создание удобного и интуитивно понятного интерфейса, который позволил бы пользователю быстро ориентироваться в системе и эффективно находить специалистов.

Процесс разработки включал решение нескольких технических задач, таких как проектирование и интеграция базы данных для хранения информации о специалистах и заказах, а также создание механизма безопасного хранения данных пользователей с использованием алгоритма хеширования паролей.

В ходе работы над проектом мы приобрели ценный опыт в области проектирования и разработки программных решений.

Не менее важным для нас был опыт работы в команде, где каждый из нас отвечал за конкретные участки работы, что позволило улучшить как технические навыки, так и навыки сотрудничества в коллективе.

В будущем планируется проведение работы над улучшением интерфейса, что повысит удобство использования и сделает его более привлекательным для пользователей. Также в планах — реализация функционала смены языка интерфейса, что позволит расширить аудиторию приложения и сделать его доступным для пользователей, говорящих на разных языках.

Одним из важных направлений для улучшения является повышение кросс-платформенности приложения, что позволит пользователям использовать его не только на настольных ПК, но и на мобильных устройствах, расширив тем самым доступность и удобство. Кроме того, мы планируем расширить функционал системы, добавив новые возможности, такие как онлайн-оплата услуг, интеграция с внешними сервисами для автоматического обновления данных и улучшение алгоритмов поиска специалистов.

# Литература

1. Самочадин А. В. Лекции и материалы курса “ПИ”.
2. Канер, Бах, Петтичерд. Тестирование программного обеспечения: основы, принципы и практика.
3. Анджелло, Эллисон. Основы тестирования программного обеспечения.
4. Мартин Фаулер. Refactoring: Improving the Design of Existing Code.
5. Дин Лейн. Создание технической документации: практическое руководство.
6. Карл Вигерс. Разработка требований к программному обеспечению.
7. Алан Купер. Об интерфейсе: основы проектирования взаимодействия.
8. Коберн, А. Писать эффективно: создание полезных сценариев использования.
9. Мартин Фаулер UML Distilled: Краткое руководство по стандартному языку объектного моделирования.
10. Документация PostgreSQL ([документацияPostgreSQL](https://www.postgresql.org/docs/)).
11. Документация JFrame ([документацияJFrame](https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html)).
12. Документация Swing([документацияSwing](https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html)).

# Приложения

## Приложение А. Руководство пользователя

Введение

Данная система предназначена для управления сотрудниками, заказами, услугами и доступом. Руководство поможет вам понять, как работать с функциональностью системы, исходя из вашей роли: Администратор, Менеджер или Специалист. Система предоставляет удобный интерфейс, который облегчает выполнение повседневных задач и улучшает взаимодействие между участниками процесса.

Роли и доступ

1. Администратор

* Полный доступ к системе.
* Управление пользователями, заказами, услугами.
* Настройка ролей и прав доступа.

2. Менеджер

* Управление заказами и специалистами.
* Доступ к данным сотрудников отдела.
* Создание и назначение задач.
* Контроль выполнения заказов.

3. Специалист

* Выполнение назначенных заказов.
* Изменение статусов заказов.
* Подготовка отчетов о проделанной работе.
* Просмотр списка доступных задач.

Основные функции системы

1. Вход в систему

* Откройте приложение.
* Введите ваш логин и пароль в форму авторизации.
* Нажмите кнопку "Войти".

2. Регистрация (только для новых пользователей)

* На экране входа нажмите кнопку "Зарегистрироваться".
* Заполните форму регистрации:
  + Логин и пароль (система проверит соответствие требованиям).
* Нажмите "Зарегистрироваться".
  + Если пароль не подходит, система покажет сообщение об ошибке с указанием требований.
  + Повторите ввод данных, если требуется.

3. Управление сотрудниками (для Администратора)

Добавление нового сотрудника

1. Перейдите в раздел "Пользователи".
2. Нажмите "Добавить сотрудника".
3. Заполните форму:
   * Имя, фамилия, должность, отдел, контактные данные.
   * Загрузите фотографию (по желанию).
4. Нажмите "Сохранить".

Редактирование сотрудника

1. Перейти на вкладку изменить список пользователей
2. Выбрать пользователя
3. Назначить роль сотрудника
4. Назначить ФИО

Удаление сотрудника

1. Найдите сотрудника в списке.
2. Нажмите "Удалить".
3. Подтвердите удаление.

4. Управление заказами

Создание нового заказа

1. Перейдите в раздел "Заказы".
2. Нажмите "Добавить заказ".
3. Заполните форму:
   * Номер услуги
   * Номер работника
   * Улица
   * Номер дома
   * Номер квартиры (опционально)
4. Нажмите "Сохранить".

Просмотр заказов

1. В разделе "Главного меню” выберите просмотр заказов).
2. Выберите заказ для просмотра подробной информации.

Изменение данных заказа

1. Откройте нужный заказ.
2. Нажмите "Редактировать".
3. Измените данные (например, сроки или описание).
4. Сохраните изменения.

Удаление заказа

1. Найдите заказ в списке.
2. Нажмите "Удалить".
3. Подтвердите удаление.

5. Управление услугами

Добавление новой услуги

1. Перейдите в раздел "Услуги".
2. Нажмите "Добавить услугу".
3. Введите данные:
   * Название.
   * Описание.
   * Стоимость.
4. Нажмите "Сохранить".

Редактирование услуги

1. Найдите услугу в списке.
2. Нажмите "Редактировать".
3. Внесите изменения и сохраните.

Удаление услуги

1. Найдите услугу в списке.
2. Нажмите "Удалить".
3. Подтвердите удаление.

6. Работа со статусами заказов (для специалистов)

Изменение статуса

1. Перейдите в раздел "Мои заказы".
2. Найдите нужный заказ.
3. Выберите статус:
   * "Выполняется".
   * "Завершен".
4. При завершении заказа отправьте отчет (если требуется).

8. Управление ролями (для Администратора)

Назначение ролей

1. Перейдите в раздел "Пользователи".
2. Найдите нужного пользователя.
3. Откройте профиль и выберите роль (Администратор, Менеджер, Специалист, Пользователь).
4. Сохраните изменения.

## Приложение Б. Сoding standard

**1. Именование**

1.1. **Классы:**

* Использовать *PascalCase*.
* Название должно отражать назначение класса (например, Application, HashUtil).

1.2. **Методы:**

* Использовать *camelCase*.
* Названия должны быть глаголами, описывающими выполняемую задачу (например, createLoginPanel, logToAccount).

1.3. **Переменные:**

* Использовать *camelCase* для переменных.
* Названия должны быть описательными:
  + Хорошо: mainPanel, userId.
  + Плохо: mp, uid.

1.4. **Константы:**

* Использовать верхний регистр с разделением через \_ (например, DEFAULT\_WINDOW\_WIDTH).

1.5. **SQL-запросы:**

* Для запросов использовать осмысленные имена строковых переменны:

String userCheck = "SELECT 1 FROM users WHERE login = ?";

**2. Форматирование**

2.1. **Отступы:**

* Использовать отступ в 4 пробела.
* Не использовать табуляцию.

2.2. **Длина строки:**

* Ограничить длину строки до 120 символов. Если строка длиннее, переносить код:
* String query = "SELECT id, login FROM users WHERE login = ? AND is\_admin = ?";

2.3. **Пробелы:**

* Добавлять пробелы между ключевыми словами и скобками:
* if (isAdmin) {
* showAdminPanel();
* }

2.4. **Блоки кода:**

* Оборачивать тело любых конструкций (if, for, while) в фигурные скобки, даже если оно состоит из одной строки:

if (isAdmin) {

JOptionPane.*showMessageDialog*(null, "Connected as administrator with login " + *userName*);

showAdminPanel();

} else {

checkSpecialistStatus(login);

}

**3. Комментарии и документация**

3.1. **Комментарии для классов и методов:**

* Добавлять описание для всех публичных классов и методов:

//создание аккаунта пользователя

private void createAccount(){…}

//помощник создания кнопок в окнах

private JButton createLoginButton(String text) {…}

3.2. **Встроенные комментарии:**

* Использовать комментарии для пояснения сложных блоков кода:
* // Проверяем, есть ли логин в базе данных
* String query = "SELECT 1 FROM users WHERE login = ?";

**4. Организация кода**

4.1. **Порядок в классе:**

* Поля класса.
* Конструкторы.
* Публичные методы.
* Приватные методы.

4.3. **Универсальность:**

* Общие методы (например, при создании интерфейса) выносить в отдельные утилитарные классы или же разделять метод на несколько других, каждый из которых будет выполнять уникальную задачу.

**5. Работа с базой данных**

5.1. **Работа исключительно по запросам:**

* Всегда работать исключительно по запросам:

buttonPanel.add(createStyledButton("View the list of services", e -> viewData("Services", "select \* from services ORDER BY id ASC")));  
buttonPanel.add(Box.*createVerticalStrut*(15));  
buttonPanel.add(createStyledButton("View the list of specialists", e -> viewData("Specialists", "select \* from specialists ORDER BY id ASC")));  
buttonPanel.add(Box.*createVerticalStrut*(15));  
buttonPanel.add(createStyledButton("Unfulfilled orders", e ->  
 viewData("Not completed", "select service\_id as \"Service id\", specialist\_id as \"Specialist id\", request\_date as \"Request date\" from service\_requests")));

5.2. **Проверка результатов:**

* Проверять возвращаемые значения и если возникают ошибки выводить окно с пояснением ошибки.

**6. Стандарты интерфейса (GUI)**

6.1. **Общие стили:**

* Придерживаться единого визуального оформления кнопок, панелей, шрифтов.

6.2. **Взаимодействие с интерфейсом:**

* Внедрить интерактивное взаимодействие с кнопками, такое как анимация при наведении и нажатии, созданное благодаря (методы animateHoverEffect и animateClickEffect)

6.3. **Кастомные элементы:**

* Для повторяющихся элементов (например, кнопок) создавать вспомогательные методы:
* private JButton createStyledButton(String text, ActionListener action) { ... }

**7. Безопасность**

7.1. **Хэширование паролей:**

* Всегда хэшировать пароли перед сохранением в базу данных с использованием BCrypt.

7.2. **Проверка ввода:**

* Проверять данные, введенные пользователем, для предотвращения ошибок и уязвимостей:
  + Убедиться, что логин и пароль не пусты.

**Преимущества стандарта**

1. **Единообразие кода:** Код становится легко читаемым и поддерживаемым.
2. **Повышенная безопасность:** Соблюдаются правила работы с паролями и базой данных.
3. **Модульность:** Легко добавлять новые функции и компоненты.

## Приложение В. Фрагменты текстов программ

Файл Main.java  
package apartmentAssistent;  
  
import javax.swing.\*;  
import java.sql.SQLException;  
  
public class Main {  
  
 public static void main(String[] args) throws SQLException {  
 SwingUtilities.*invokeLater*(() -> {  
 Application app = null;  
 try {  
 app = new Application();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 app.setVisible(true);  
 });  
 }  
}

Файл HashUtil.java

package apartmentAssistent;  
  
import org.mindrot.jbcrypt.BCrypt;  
  
public class HashUtil {  
 public static String hash(String getIt) {  
 return BCrypt.*hashpw*(getIt, BCrypt.*gensalt*());  
 }  
  
 public static boolean check(String getIt, String hashedInfo) {  
 return BCrypt.*checkpw*(getIt, hashedInfo);  
 }  
}

Файл Config.java

package apartmentAssistent;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.Properties;  
  
public class Config {  
 public static Properties loadProperties() {  
 Properties properties = new Properties();  
 try (InputStream input = Config.class.getClassLoader().getResourceAsStream("config.properties")) {  
 if (input == null) {  
 System.*out*.println("Sorry, couldn't find config.properties");  
 return properties;  
 }  
 properties.load(input);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println(properties);  
 return properties;  
 }  
}

Пример файла Config.properties

database.url=jdbc:postgresql://localhost:1111/apartament\_helper\_db  
  
database.login=postgres  
  
database.password=stud

Файл build.gradle

plugins **{** id 'java'  
**}**group = 'org.example'  
version = '1.0-SNAPSHOT'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** implementation 'org.mindrot:jbcrypt:0.4'  
 implementation 'org.postgresql:postgresql:42.6.0'  
 implementation'org.apache.poi:poi-ooxml:5.2.3'  
 implementation 'org.apache.logging.log4j:log4j-core:2.20.0'  
**}**tasks.withType(JavaCompile) **{** options.encoding = 'UTF-8'  
**}**tasks.named('processResources') **{** duplicatesStrategy = DuplicatesStrategy.*EXCLUDE***}**sourceSets **{** main **{** resources **{** srcDir 'src/main/resources'  
 **}  
 }  
}**

Фрагменты файла Application.java, посвященные регистрации, создании окна и методов специалиста

package apartmentAssistent;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.awt.image.BufferedImage;  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.Properties;  
  
public class Application extends JFrame {  
 private JPanel mainPanel;  
 private CardLayout cardLayout;  
 private Connection connection;  
  
 static String *userName*;  
 boolean isAdmin = false;  
 static int *userId*;  
 static int *specialistId*;  
  
 public Application() throws SQLException {  
 super("Welcome to Apartment Assistant");  
 this.setSize(430, 610);  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.setLocationRelativeTo(null);  
  
 mainPanel = new JPanel(new CardLayout());  
 cardLayout = (CardLayout) mainPanel.getLayout();  
  
 JPanel loginPanel = createLoginPanel();  
 mainPanel.add(loginPanel, "LoginPanel");  
 this.add(mainPanel);  
  
 Properties config = Config.*loadProperties*();  
 this.connection = DriverManager.*getConnection*(  
 config.getProperty("database.url"),  
 config.getProperty("database.login"),  
 config.getProperty("database.password"));  
 }  
   
 private JButton createLoginButton(String text) {  
 JButton button = new JButton(text) {  
 private final int dotSize = 2;  
 private final int spacing = 1;  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 Graphics2D g2d = (Graphics2D) g;  
 g2d.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*, RenderingHints.*VALUE\_ANTIALIAS\_ON*);  
  
 int arcWidth = 30;  
 int arcHeight = 30;  
  
 g2d.setColor(Color.*BLACK*);  
 g2d.fillRoundRect(5, 5, getWidth() - 5, getHeight() - 5, arcWidth, arcHeight);  
  
 GradientPaint gradient = new GradientPaint(  
 0, 0, new Color(118, 72, 255),  
 getWidth(), getHeight(), new Color(23, 0, 122)  
 );  
 g2d.setPaint(gradient);  
 g2d.fillRoundRect(0, 0, getWidth(), getHeight(), arcWidth, arcHeight);  
  
 g2d.setColor(new Color(30, 0, 63));  
 g2d.setStroke(new BasicStroke(4));  
 g2d.drawRoundRect(1, 1, getWidth() - 3, getHeight() - 3, arcWidth, arcHeight);  
  
 Color startColor = new Color(30, 0, 63, 0);  
 Color endColor = new Color(30, 0, 63, 150);  
  
 for (int i = 0; i < getWidth(); i += dotSize + spacing) {  
 for (int j = 0; j < getHeight(); j += dotSize + spacing) {  
 int x = i + j;  
 int y = j;  
  
 if (x < getWidth()) {  
 if (isPointInsideRoundedRect(x, y, getWidth(), getHeight(), arcWidth, arcHeight)) {  
 float fraction = (float) x / getWidth();  
 Color interpolatedColor = interpolateColor(startColor, endColor, fraction);  
 g2d.setColor(interpolatedColor);  
 g2d.fillOval(x, y, dotSize, dotSize);  
 }  
 }  
 }  
 }  
 super.paintComponent(g);  
 }  
  
 private boolean isPointInsideRoundedRect(int x, int y, int width, int height, int arcWidth, int arcHeight) {  
 java.awt.geom.RoundRectangle2D roundedRect = new java.awt.geom.RoundRectangle2D.Float(  
 0, 0, width, height, arcWidth, arcHeight  
 );  
 return roundedRect.contains(x, y);  
 }  
  
 private Color interpolateColor(Color butt\_color1, Color butt\_color2, float fraction) {  
 float r = butt\_color1.getRed() + (butt\_color2.getRed() - butt\_color1.getRed()) \* fraction;  
 float g = butt\_color1.getGreen() + (butt\_color2.getGreen() - butt\_color1.getGreen()) \* fraction;  
 float b = butt\_color1.getBlue() + (butt\_color2.getBlue() - butt\_color1.getBlue()) \* fraction;  
 float a = butt\_color1.getAlpha() + (butt\_color2.getAlpha() - butt\_color1.getAlpha()) \* fraction;  
 return new Color((int) r, (int) g, (int) b, (int) a);  
 }  
 };  
  
 button.setFocusPainted(false);  
 button.setContentAreaFilled(false);  
 button.setBorderPainted(false);  
 button.setOpaque(false);  
 button.setFont(new Font("JetBrains Mono", Font.*PLAIN*, 28));  
 button.setForeground(Color.*WHITE*);  
 button.setHorizontalAlignment(SwingConstants.*CENTER*);  
 button.setPreferredSize(new Dimension(550, 100));  
  
 button.addActionListener(e -> {  
 if (text.equals("LOG IN")) {  
 logToAccount();  
 } else if (text.equals("SIGN UP")) {  
 createAccount();  
 }  
 });  
  
 return button;  
 }  
  
  
 // самое первое окно, в котором пользователь входит\создает ак  
 private JPanel createLoginPanel() {  
 JPanel panel = new JPanel(new BorderLayout()) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 Graphics2D g2d = (Graphics2D) g;  
 Color color1 = new Color(255, 0, 212);  
 Color color2 = new Color(122, 0, 255);  
 Color color3 = new Color(255, 122, 236);  
 Color color4 = new Color(0, 3, 255);  
  
 GradientPaint gradient = new GradientPaint(0, 0, color1, getWidth(), getHeight(), color2);  
 g2d.setPaint(gradient);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 BufferedImage noiseImage = new BufferedImage(getWidth(), getHeight(), BufferedImage.*TYPE\_INT\_ARGB*);  
 for (int x = 0; x < getWidth(); x++) {  
 for (int y = 0; y < getHeight(); y++) {  
 int noise = (int) (Math.*random*() \* 50);  
 int alpha = 60;  
 int rgba = new Color(noise, noise, noise, alpha).getRGB();  
 noiseImage.setRGB(x, y, rgba);  
 }  
 }  
  
 g2d.drawImage(noiseImage, 0, 0, null);  
  
 int dotSize = 2;  
 int spacing = 3;  
  
 for (int x = 0; x < getWidth(); x += spacing) {  
 for (int y = 0; y < getHeight(); y += spacing) {  
 float fraction = (float) (x + y) / (getWidth() + getHeight());  
 Color dotColor = interpolateColor(color3, color4, fraction);  
 g2d.setColor(dotColor);  
 g2d.fillOval(x, y, dotSize, dotSize);  
 }  
 }  
  
 int secondDotSize = 12;  
 int secondDotSpacing = 7;  
  
 float fadeOutFactor = 2.2f;  
  
 for (int x = 0; x < getWidth(); x += secondDotSpacing) {  
 for (int y = 0; y < getHeight(); y += secondDotSpacing) {  
 float distanceFraction = (float) (x + y) / (fadeOutFactor \* (getWidth() + getHeight()));  
 distanceFraction = Math.*min*(distanceFraction, 1);  
 int alpha = (int) (255 \* distanceFraction);  
 alpha = Math.*max*(alpha, 0);  
  
 Color dotColor = new Color(0, 2, 20, alpha);  
 g2d.setColor(dotColor);  
 g2d.fillOval(x, y, secondDotSize, secondDotSize);  
 }  
 }  
 }  
 private Color interpolateColor(Color color3, Color color4, float fraction) {  
 float r = color3.getRed() + (color4.getRed() - color3.getRed()) \* fraction;  
 float g = color3.getGreen() + (color4.getGreen() - color3.getGreen()) \* fraction;  
 float b = color3.getBlue() + (color4.getBlue() - color3.getBlue()) \* fraction;  
 return new Color((int) r, (int) g, (int) b);  
 }  
 };  
  
 panel.setPreferredSize(new Dimension(450, 600));  
 panel.setBorder(BorderFactory.*createEmptyBorder*(20, 20, 20, 20));  
  
 JPanel centerContainer = new JPanel();  
 centerContainer.setLayout(new BoxLayout(centerContainer, BoxLayout.*Y\_AXIS*));  
 centerContainer.setOpaque(false);  
  
 JLabel label = new JLabel("WELCOME!", JLabel.*CENTER*);  
 label.setFont(new Font("JetBrains Mono", Font.*PLAIN*, 38));  
 label.setForeground(Color.*WHITE*);  
 label.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
  
 label.setBorder(BorderFactory.*createCompoundBorder*(  
 BorderFactory.*createEmptyBorder*(4, 4, 0, 0),  
 BorderFactory.*createLineBorder*(Color.*BLACK*, 0)  
 ));  
  
 centerContainer.add(label);  
  
 centerContainer.add(Box.*createRigidArea*(new Dimension(0, 140)));  
  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.*Y\_AXIS*));  
 buttonPanel.setOpaque(false);  
  
 JButton loginButton = createLoginButton("LOG IN");  
 loginButton.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 buttonPanel.add(loginButton);  
 buttonPanel.add(Box.*createRigidArea*(new Dimension(5, 20))); // Smaller gap between buttons  
  
 JButton createAccountButton = createLoginButton("SIGN UP");  
 createAccountButton.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 buttonPanel.add(createAccountButton);  
  
 centerContainer.add(buttonPanel);  
  
 panel.add(centerContainer, BorderLayout.*CENTER*);  
  
 return panel;  
 }  
  
 //создание аккаунта пользователя  
 private void createAccount() {  
 JFrame logFrame = new JFrame("SIGN UP");  
 logFrame.setSize(400, 200);  
 logFrame.setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 logFrame.setLocationRelativeTo(this);  
  
 Font customFont = new Font("JetBrains Mono", Font.*PLAIN*, 14);  
  
 JPanel inputPanel = new JPanel(new GridBagLayout()) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 Graphics2D g2d = (Graphics2D) g;  
 g2d.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*, RenderingHints.*VALUE\_ANTIALIAS\_ON*);  
  
 GradientPaint gradient = new GradientPaint(  
 0, 0, new Color(148, 0, 255),  
 getWidth(), getHeight(), new Color(0, 0, 0)  
 );  
 g2d.setPaint(gradient);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 g2d.setColor(new Color(255, 133, 245, 50));  
 int dotSize = 2;  
 int spacing = 2;  
  
 for (int x = 0; x < getWidth(); x += spacing) {  
 for (int y = 0; y < getHeight(); y += spacing) {  
 g2d.fillOval(x, y, dotSize, dotSize);  
 }  
 }  
 }  
 };  
  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.insets = new Insets(5, 5, 5, 5);  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
  
 JLabel loginLabel = new JLabel("Login:");  
 loginLabel.setForeground(Color.*WHITE*);  
 loginLabel.setFont(customFont);  
 JLabel passwordLabel = new JLabel("Password:");  
 passwordLabel.setForeground(Color.*WHITE*);  
 passwordLabel.setFont(customFont);  
  
 JTextField loginField = new JTextField(15);  
 loginField.setFont(customFont);  
 JPasswordField passwordField = new JPasswordField(15);  
 passwordField.setFont(customFont);  
  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.anchor = GridBagConstraints.*EAST*;  
 inputPanel.add(loginLabel, gbc);  
  
 gbc.gridx = 1;  
 gbc.anchor = GridBagConstraints.*WEST*;  
 inputPanel.add(loginField, gbc);  
  
 gbc.gridx = 0;  
 gbc.gridy = 1;  
 gbc.anchor = GridBagConstraints.*EAST*;  
 inputPanel.add(passwordLabel, gbc);  
  
 gbc.gridx = 1;  
 gbc.anchor = GridBagConstraints.*WEST*;  
 inputPanel.add(passwordField, gbc);  
  
 JButton addButton = new JButton("Sign Up");  
 addButton.setFont(customFont); // Apply font  
 addButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String login = loginField.getText();  
 String password = new String(passwordField.getPassword());  
  
 if (login.trim().isEmpty() || password.trim().isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "Fill in all the fields");  
 return;  
 }  
 try {  
 if (checkLoginExists(login)) {  
 JOptionPane.*showMessageDialog*(null, "User with this login already exists!");  
 return;  
 }  
 PreparedStatement statement = connection.prepareStatement(  
 "INSERT INTO users (login, password) VALUES(?,?)"  
 );  
 statement.setString(1, login);  
 statement.setString(2, HashUtil.*hash*(password));  
 int rowsAffected = statement.executeUpdate();  
 if (rowsAffected > 0) {  
 JOptionPane.*showMessageDialog*(null, "The account has been successfully created!");  
 logFrame.dispose();  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Error occurred when creating an account", "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Error occurred when creating an account: " + ex.getMessage(), "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 });  
  
 JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*));  
 buttonPanel.add(addButton);  
  
 logFrame.add(inputPanel, BorderLayout.*CENTER*); // Place inputPanel in the center  
 logFrame.add(buttonPanel, BorderLayout.*SOUTH*);  
 logFrame.setVisible(true);  
 }  
  
 //проверка наличия акка по логину  
 private boolean checkLoginExists(String login) throws SQLException {  
 String sql = "SELECT 1 FROM users WHERE login = ?";  
 try (PreparedStatement statement = connection.prepareStatement(sql)) {  
 statement.setString(1, login);  
 ResultSet rs = statement.executeQuery();  
 return rs.next();  
 }  
 }  
  
 //вход в ак по логину и паролю  
 private void logToAccount() {  
 JFrame logFrame = new JFrame("LOG IN");  
 logFrame.setSize(400, 200);  
 logFrame.setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 logFrame.setLocationRelativeTo(this);  
  
 Font customFont = new Font("JetBrains Mono", Font.*PLAIN*, 14);  
  
 JPanel inputPanel = new JPanel(new GridBagLayout()) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 Graphics2D g2d = (Graphics2D) g;  
  
 g2d.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*, RenderingHints.*VALUE\_ANTIALIAS\_ON*);  
  
 GradientPaint gradient = new GradientPaint(  
 0, 0, new Color(0, 13, 82),  
 getWidth(), getHeight(), new Color(255, 0, 144)  
 );  
 g2d.setPaint(gradient);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 g2d.setColor(new Color(153, 255, 142, 113));  
 int dotSize = 1;  
 int spacing = 2;  
  
 for (int x = 0; x < getWidth(); x += spacing) {  
 for (int y = 0; y < getHeight(); y += spacing) {  
 g2d.fillOval(x, y, dotSize, dotSize);  
 }  
 }  
 }  
 };  
  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.insets = new Insets(5, 5, 5, 5);  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
  
 JLabel loginLabel = new JLabel("Login:");  
 loginLabel.setForeground(Color.*WHITE*);  
 loginLabel.setFont(customFont);  
 JLabel passwordLabel = new JLabel("Password:");  
 passwordLabel.setForeground(Color.*WHITE*);  
 passwordLabel.setFont(customFont);  
  
 JTextField loginField = new JTextField(15);  
 loginField.setFont(customFont);  
 JPasswordField passwordField = new JPasswordField(15);  
 passwordField.setFont(customFont);  
  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.anchor = GridBagConstraints.*EAST*;  
 inputPanel.add(loginLabel, gbc);  
  
 gbc.gridx = 1;  
 gbc.anchor = GridBagConstraints.*WEST*;  
 inputPanel.add(loginField, gbc);  
  
 gbc.gridx = 0;  
 gbc.gridy = 1;  
 gbc.anchor = GridBagConstraints.*EAST*;  
 inputPanel.add(passwordLabel, gbc);  
  
 gbc.gridx = 1;  
 gbc.anchor = GridBagConstraints.*WEST*;  
 inputPanel.add(passwordField, gbc);  
  
 JButton addButton = new JButton("Log In");  
 addButton.setFont(customFont);  
 addButton.addActionListener(e -> {  
 String login = loginField.getText().trim();  
 String password = new String(passwordField.getPassword()).trim();  
 if (login.isEmpty() || password.isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "Fill in all the fields", "Input error", JOptionPane.*ERROR\_MESSAGE*);  
 return;  
 }  
 try {  
 String checkUserQuery = "SELECT password, is\_admin FROM users WHERE login = ?";  
 try (PreparedStatement checkUserStatement = connection.prepareStatement(checkUserQuery)) {  
 checkUserStatement.setString(1, login);  
  
 try (ResultSet userResult = checkUserStatement.executeQuery()) {  
 if (userResult.next()) {  
 String storedPassword = userResult.getString("password");  
 if (HashUtil.*check*(password, storedPassword)) {  
 isAdmin = userResult.getBoolean("is\_admin");  
 *userName* = login;  
 logFrame.setVisible(false);  
 if (isAdmin) {  
 JOptionPane.*showMessageDialog*(null, "Connected as administrator with login " + *userName*);  
 showAdminPanel();  
 } else {  
 checkSpecialistStatus(login);  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Invalid password", "Input error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(null, "No account found with this login", "Logging error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 }  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Error when logging into the account: " + ex.getMessage(), "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 });  
  
 JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*));  
 buttonPanel.add(addButton);  
  
 logFrame.add(inputPanel, BorderLayout.*CENTER*);  
 logFrame.add(buttonPanel, BorderLayout.*SOUTH*);  
 logFrame.setVisible(true);  
 }  
  
 //проверка специалист или пользователь  
 private void checkSpecialistStatus(String login) {  
 try {  
 String findWorkerQuery = "SELECT id FROM specialists WHERE lower(login) = lower(?)";  
 login = login.trim();  
 System.*out*.println("Final login value (trimmed): '" + login + "'");  
 try (PreparedStatement findWorkerStatement = connection.prepareStatement(findWorkerQuery)) {  
 findWorkerStatement.setString(1, login);  
 try (ResultSet specialistResult = findWorkerStatement.executeQuery()) {  
 if (specialistResult.next()) {  
 *specialistId* = specialistResult.getInt("id");  
 JOptionPane.*showMessageDialog*(null, "Connected as specialist with login " + login + " (ID: " + *specialistId* + ")");  
 showSpecialistPanel();  
 } else {  
 try (PreparedStatement findUserIdStatement = connection.prepareStatement(  
 "SELECT id FROM users WHERE login = ?")) {  
 findUserIdStatement.setString(1, login);  
 try (ResultSet rs = findUserIdStatement.executeQuery()) {  
 if (rs.next()) {  
 *userId* = rs.getInt("id");  
 System.*out*.println(*userId*);  
 JOptionPane.*showMessageDialog*(null, "Connected as user with login " + login + " (ID: " + *userId* + ")");  
 showUserPanel();  
 } else {  
 JOptionPane.*showMessageDialog*(null, "No user found with login " + login);  
 }  
 }  
 } catch (SQLException ex) {  
 JOptionPane.*showMessageDialog*(null, "Database error while finding user ID");  
 ex.printStackTrace();  
 }  
 showUserPanel();  
 }  
 }  
 }  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Error when verifying specialist: " + ex.getMessage(), "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 //добавление окон для специалиста  
 private void showSpecialistPanel(){  
 mainPanel.removeAll();  
 JPanel loginPanel =createLoginPanel();  
 mainPanel.add(loginPanel,"Login Panel");  
 JPanel mainSpecialistPanel=mainSpecialistPanel();  
 mainPanel.add(mainSpecialistPanel, "Main Specialist Panel");  
 cardLayout.show(mainPanel,"Main Specialist Panel");  
 }  
  
 //добавление окон для админа  
 private void showAdminPanel() {  
 mainPanel.removeAll();  
 JPanel loginPanel = createLoginPanel();  
 mainPanel.add(loginPanel, "LoginPanel");  
 JPanel mainAdminPanel = mainAdminPanel();  
 mainPanel.add(mainAdminPanel, "MainAdminPanel");  
 cardLayout.show(mainPanel, "MainAdminPanel");  
 }  
  
 //добавление окон для пользователя  
 private void showUserPanel() {  
 mainPanel.removeAll();  
 JPanel loginPanel = createLoginPanel();  
 mainPanel.add(loginPanel, "LoginPanel");  
 JPanel mainUserPanel = mainUserPanel();  
 mainPanel.add(mainUserPanel, "MainUserPanel");  
 cardLayout.show(mainPanel, "MainUserPanel");  
 }  
  
 //главное меню специалиста  
 private JPanel mainSpecialistPanel() {  
 JPanel mainPanel = new JPanel();  
 mainPanel.setLayout(new BorderLayout());  
 mainPanel.setBackground(new Color(230, 230, 250));  
  
 JPanel backgroundPanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 Graphics2D g2d = (Graphics2D) g.create();  
  
 GradientPaint gradient = new GradientPaint(0, 0, new Color(255, 77, 117),  
 getWidth(), getHeight(), new Color(131, 0, 248));  
 g2d.setPaint(gradient);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 int dotSize = 5;  
 int spacing = 2;  
 for (int x = 0; x < getWidth(); x += spacing) {  
 for (int y = 0; y < getHeight(); y += spacing) {  
 g2d.setColor(new Color(249, 168, 255, 60));  
 g2d.fillOval(x, y, dotSize, dotSize);  
 }  
 }  
  
 g2d.dispose();  
 }  
 };  
 backgroundPanel.setLayout(null);

JLabel titleLabel = new JLabel("Hello! Welcome to Assistant!", JLabel.*CENTER*) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 Graphics2D g2d = (Graphics2D) g.create();  
 String text = getText();  
 FontMetrics fm = g2d.getFontMetrics(getFont());  
  
 int x = (getWidth() - fm.stringWidth(text)) / 2;  
 int y = (getHeight() + fm.getAscent()) / 2;  
   
 g2d.setColor(new Color(0, 0, 0, 150));  
 g2d.drawString(text, x + 2, y + 2);  
 g2d.setColor(Color.*BLACK*);  
 g2d.drawString(text, x - 1, y - 1);  
 g2d.drawString(text, x + 1, y - 1);  
 g2d.drawString(text, x - 1, y + 1);  
 g2d.drawString(text, x + 1, y + 1);  
 g2d.setColor(Color.*WHITE*);  
 g2d.drawString(text, x, y);  
 g2d.dispose();  
 }  
 };  
 titleLabel.setFont(new Font("JetBrains Mono", Font.*BOLD* | Font.*ITALIC*, 25));  
 titleLabel.setForeground(Color.*WHITE*);  
 titleLabel.setOpaque(false);  
 titleLabel.setBounds(0, 20, 800, 40);  
 backgroundPanel.add(titleLabel);  
   
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.*Y\_AXIS*));  
 buttonPanel.setOpaque(false);  
 buttonPanel.setBounds(50, 100, 500, 400);  
 buttonPanel.add(Box.*createVerticalGlue*());  
  
 buttonPanel.add(createStyledButton("View your outstanding orders", e -> viewData("Orders", "select \* from service\_requests where specialist\_id=" + *specialistId*)));  
 buttonPanel.add(Box.*createVerticalStrut*(15));  
 buttonPanel.add(createStyledButton("Send the completed order id", e -> completeOrder()));  
 buttonPanel.add(Box.*createVerticalStrut*(15));  
 buttonPanel.add(createStyledButton("View the orders you can complete", e -> {  
 String query = String.*format*(  
 "select sr.\*from service\_requests sr where sr.service\_id = any (select unnest(s.service\_id) from specialists s " +  
 "where s.id = %d)", *specialistId*);  
 viewData("Orders", query);  
 }));  
 buttonPanel.add(Box.*createVerticalStrut*(15));  
 buttonPanel.add(createStyledButton("Make an action with list of services", e -> changeSpecialistInOrder()));  
  
 buttonPanel.add(Box.*createVerticalGlue*());  
  
 backgroundPanel.add(buttonPanel);  
 mainPanel.add(backgroundPanel, BorderLayout.*CENTER*);  
  
 return mainPanel;  
}  
  
//смена специалиста в заказе  
private void changeSpecialistInOrder() {  
 JFrame changeSpecialistFrame = new JFrame("Change Specialist in Order");  
 changeSpecialistFrame.setSize(400, 200);  
 changeSpecialistFrame.setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 changeSpecialistFrame.setLocationRelativeTo(this);  
  
 JPanel inputPanel = new JPanel(new GridLayout(2, 1, 10, 10));  
 JTextField requestIdField = new JTextField();  
  
 inputPanel.add(new JLabel("Enter the ID of the service request (must be > 0):"));  
 inputPanel.add(requestIdField);  
  
 JButton updateButton = new JButton("Update Specialist");  
 updateButton.addActionListener(e -> {  
 String requestIdText = requestIdField.getText().trim();  
 if (requestIdText.isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "Please enter a valid request ID");  
 return;  
 }  
 int requestId;  
 try {  
 requestId = Integer.*parseInt*(requestIdText);  
 if (requestId <= 0) {  
 JOptionPane.*showMessageDialog*(null, "The request ID must be greater than 0");  
 return;  
 }  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(null, "The request ID must be a numeric value");  
 return;  
 }  
  
 try {  
 connection.setAutoCommit(false);  
 int serviceId = -1;  
 try (PreparedStatement checkRequestStatement = connection.prepareStatement(  
 "SELECT service\_id, specialist\_id FROM service\_requests WHERE id = ?")) {  
 checkRequestStatement.setInt(1, requestId);  
 try (ResultSet rs = checkRequestStatement.executeQuery()) {  
 if (rs.next()) {  
 serviceId = rs.getInt("service\_id");  
 Integer currentSpecialistId = rs.getObject("specialist\_id", Integer.class);  
 if (currentSpecialistId != null) {  
 JOptionPane.*showMessageDialog*(null, "This order already has a specialist assigned.");  
 return;  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(null, "No service request found with the given ID.");  
 return;  
 }  
 }  
 }  
  
 boolean isServiceSupported = false;  
 try (PreparedStatement checkSpecialistStatement = connection.prepareStatement(  
 "SELECT 1 FROM specialists WHERE id = ? AND ? = ANY (service\_id)")) {  
 checkSpecialistStatement.setInt(1, *specialistId*);  
 checkSpecialistStatement.setInt(2, serviceId);  
  
 try (ResultSet rs = checkSpecialistStatement.executeQuery()) {  
 if (rs.next()) {  
 isServiceSupported = true;  
 }  
 }  
 }  
  
 if (!isServiceSupported) {  
 JOptionPane.*showMessageDialog*(null, "The specialist cannot complete this service.");  
 return;  
 }  
  
 try (PreparedStatement updateStatement = connection.prepareStatement(  
 "UPDATE service\_requests SET specialist\_id = ? WHERE id = ?")) {  
 updateStatement.setInt(1, *specialistId*);  
 updateStatement.setInt(2, requestId);  
 int rowsUpdated = updateStatement.executeUpdate();  
 if (rowsUpdated > 0) {  
 JOptionPane.*showMessageDialog*(null, "Specialist has been successfully assigned to the order.");  
 connection.commit();  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Failed to update the order.");  
 connection.rollback();  
 }  
 }  
 } catch (SQLException ex) {  
 JOptionPane.*showMessageDialog*(null, "Database connection error");  
 ex.printStackTrace();  
 try {  
 connection.rollback();  
 } catch (SQLException rollbackEx) {  
 rollbackEx.printStackTrace();  
 }  
 } finally {  
 try {  
 connection.setAutoCommit(true);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 });  
  
 JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*));  
 buttonPanel.add(updateButton);  
 changeSpecialistFrame.add(inputPanel, BorderLayout.*CENTER*);  
 changeSpecialistFrame.add(buttonPanel, BorderLayout.*SOUTH*);  
 changeSpecialistFrame.setVisible(true);  
}  
  
//отобразить, что заказ выполнен  
private void completeOrder() {  
 String input = JOptionPane.*showInputDialog*(null,  
 "Enter the ID of the order to complete (must be > 0):",  
 "Complete Order",  
 JOptionPane.*QUESTION\_MESSAGE*);  
  
 if (input == null) {  
 JOptionPane.*showMessageDialog*(null, "Action canceled.", "Info", JOptionPane.*INFORMATION\_MESSAGE*);  
 return;  
 }  
  
 try {  
 int orderId = Integer.*parseInt*(input.trim());  
 if (orderId <= 0) {  
 JOptionPane.*showMessageDialog*(null,  
 "The ID must be a positive number greater than 0.",  
 "Invalid Input",  
 JOptionPane.*ERROR\_MESSAGE*);  
 return;  
 }  
  
 String requestDate = null;  
 int serviceId = -1;  
 try (PreparedStatement findStmt = connection.prepareStatement(  
 "SELECT request\_date, service\_id FROM service\_requests WHERE id = ?")) {  
 findStmt.setInt(1, orderId);  
 try (ResultSet rs = findStmt.executeQuery()) {  
 if (rs.next()) {  
 requestDate = rs.getDate("request\_date").toString(); // Преобразуем дату в строку  
 serviceId = rs.getInt("service\_id");  
 } else {  
 JOptionPane.*showMessageDialog*(null,  
 "No order with ID " + orderId + " was found.",  
 "Order Not Found",  
 JOptionPane.*WARNING\_MESSAGE*);  
 return;  
 }  
 }  
 }  
  
 int confirm = JOptionPane.*showConfirmDialog*(null,  
 "Order Details:\nRequest Date: " + requestDate + "\nService ID: " + serviceId +  
 "\n\nDo you really want to delete this order?",  
 "Confirm Deletion",  
 JOptionPane.*YES\_NO\_OPTION*,  
 JOptionPane.*WARNING\_MESSAGE*);  
 if (confirm != JOptionPane.*YES\_OPTION*) {  
 JOptionPane.*showMessageDialog*(null, "Deletion canceled.", "Info", JOptionPane.*INFORMATION\_MESSAGE*);  
 return;  
 }  
  
 try (PreparedStatement deleteStmt = connection.prepareStatement("DELETE FROM service\_requests WHERE id = ?")) {  
 deleteStmt.setInt(1, orderId);  
 int rowsAffected = deleteStmt.executeUpdate();  
 if (rowsAffected > 0) {  
 JOptionPane.*showMessageDialog*(null,  
 "Order with ID " + orderId + " was successfully completed and removed.",  
 "Success",  
 JOptionPane.*INFORMATION\_MESSAGE*);  
 } else {  
 JOptionPane.*showMessageDialog*(null,  
 "Failed to delete order with ID " + orderId + ".",  
 "Error",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 String insertQuery = "INSERT INTO completed\_orders (order\_creation\_date, order\_completion\_date, user\_comment, service\_id, specialist\_id) VALUES (?, ?, ?, ?, ?)";  
 try (PreparedStatement insertStatement = connection.prepareStatement(insertQuery)) {  
 insertStatement.setDate(1, Date.*valueOf*(requestDate)); // order\_creation\_date  
 insertStatement.setDate(2, new Date(System.*currentTimeMillis*())); // order\_completion\_date  
 insertStatement.setString(3, "No comments (release soon)"); // user\_comment  
 insertStatement.setInt(4, serviceId); // service\_id  
 insertStatement.setInt(5, *specialistId*); // specialist\_id  
 insertStatement.executeUpdate();  
 }  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(null,  
 "Invalid input. Please enter a valid positive integer.",  
 "Invalid Input",  
 JOptionPane.*ERROR\_MESSAGE*);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(null,  
 "An error occurred while processing the order: " + ex.getMessage(),  
 "Database Error",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
}