Изображение выглядит как графическая вставка, Графика, мультфильм

Автоматически созданное описание

**КУРСОВОЙ ПРОЕКТ**

**По дисциплине: МДК 01.01 Разработка программных модулей**

**Тема: Разработка системы классов для приложения «Кадровое агентство»**

**Специальность 09.02.07 «Информационные системы и программирование»**

|  |  |  |
| --- | --- | --- |
| **Выполнил студент(ка) группы 302ИС-22** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **С.Д. Сангаджи-Горяева** |
| **Руководитель** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Л.Б. Гусятинер** |

**Москва 2024**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc185718572)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ДЕЯТЕЛЬНОСТИ 4](#_Toc185718573)

[1.1 Информационное обеспечение задачи 4](#_Toc185718574)

[1.2 Обзор и анализ существующих программных решений 5](#_Toc185718575)

[1.3 Постановка задачи. Структура входной и выходной информации 7](#_Toc185718576)

[2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА СИСТЕМЫ КЛАССОВ 9](#_Toc185718577)

[2.1 Построение модели системы 9](#_Toc185718578)

[2.2 Описание среды разработки 9](#_Toc185718579)

[2.3 Построение диаграммы прецедентов и диаграммы классов 11](#_Toc185718580)

[2.4 Описание модулей 12](#_Toc185718581)

[2.5 Расчет сложности алгоритма 15](#_Toc185718582)

[3. ТЕСТИРОВАНИЕ И ОТЛАДКА СИСТЕМЫ 17](#_Toc185718583)

[3.1 Тестирование системы методом черного ящика 17](#_Toc185718584)

[3.2 Описание применения средств отладки 20](#_Toc185718585)

[ЗАКЛЮЧЕНИЕ 22](#_Toc185718586)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc185718587)

[ПРИЛОЖЕНИЕ 25](#_Toc185718588)

[Приложение 1. Блок-схемы 25](#_Toc185718589)

[Приложение 2. Код программы 28](#_Toc185718590)

[Приложение 3. Сценарий работы программы 40](#_Toc185718591)

[Приложение 4. Графический интерфейс 41](#_Toc185718592)

# ВВЕДЕНИЕ

Цифровые технологии изменяют все сферы жизни, включая управление персоналом. Кадровые агентства, играющие ключевую роль в рынке труда, сталкиваются с ростом объема данных и необходимостью автоматизации процессов подбора персонала. Автоматизация упрощает многие процессы, такие как анализ вакансий, проверка кандидатов и оформление трудовых отношений, освобождая сотрудников от рутинной работы и позволяя сосредоточиться на более важных задачах.

Практическая значимость проекта — создание системы классов для автоматизации процессов управления персоналом, что повысит эффективность работы кадровых агентств.

Актуальность проекта заключается в необходимости автоматизации процессов обработки и анализа данных в кадровых агентствах, таких как хранение информации о кандидатах, их поиск и создание отчетов.

Цель проекта — разработка системы классов для автоматизации хранения и обработки данных.

Задачи проекта включают анализ предметной области и определение ключевых задач, проектирование структуры классов и базы данных, реализацию основных функций системы, тестирование системы на различных сценариях и предложение путей расширения функционала.

Объект исследования — процессы управления информацией в кадровом агентстве. Предмет исследования — методы автоматизации этих процессов через проектирование системы классов.

# АНАЛИЗ ПРЕДМЕТНОЙ ДЕЯТЕЛЬНОСТИ

## Информационное обеспечение задачи

Кадровое агентство представляет собой специализированную организацию, которая занимается подбором и трудоустройством персонала для различных компаний. В своей деятельности кадровые агентства действуют как посредники между работодателями и соискателями, обеспечивая удовлетворение потребностей обеих сторон.

Клиенты кадрового агентства:

* Компании-работодатели. Это организации, которые обращаются в агентство для поиска сотрудников. Они предоставляют информацию о вакансиях, требованиях к кандидатам и условиях работы.
* Соискатели. Это физические лица, которые ищут работу через агентство. Они предоставляют свои резюме и участвуют в процессе подбора.

Объекты управления:

* База данных кандидатов. Содержит информацию о соискателях, включая их персональные данные, опыт работы, навыки, предпочтения и сопутствующую информацию.
* База данных вакансий. Содержит сведения о рабочих местах, которые необходимо заполнить: описание должностей, требования, условия работы, сроки подачи заявок.
* История взаимодействий. Включает данные о процессе взаимодействия с клиентами и кандидатами, таких как отправленные резюме, отклики, собеседования и итоговые результаты.

Процессы взаимодействия:

* Поиск подходящих кандидатов под вакансии. Основной процесс, включающий анализ требований работодателей и сопоставление их с информацией в базе данных соискателей.
* Ведение документооборота. Включает составление отчетов, соглашений, договоров, а также хранение данных в соответствии с законодательными требованиями.

Агентство отслеживает статусы заявок и уведомляет клиентов и соискателей. Анализ и отчетность включают генерацию отчетов и анализ эффективности. Управление контрактами и договорами завершает процесс, согласуя условия с работодателями и соискателями. Эти процессы обеспечивают высокое качество подбора персонала.

## Обзор и анализ существующих программных решений

Современные кадровые агентства активно используют программные решения для автоматизации процессов подбора персонала, управления базами данных кандидатов и работодателей, а также аналитики эффективности работы. На рынке представлено множество таких систем, каждая из которых имеет свои преимущества и особенности. Рассмотрим основные программные продукты, используемые в этой сфере, и проведем их краткий анализ.

1. HRM-системы общего назначения: многие кадровые агентства используют универсальные HRM-системы (Human Resource Management), такие как:

* 1С: Зарплата и управление персоналом

Эта система популярна благодаря своей интеграции с другими продуктами 1С, широким возможностям для учета сотрудников, расчета заработной платы и ведения баз данных. Однако она больше подходит для внутреннего использования на предприятиях, чем для специфичных задач кадровых агентств.

* SAP SuccessFactors

Мощное корпоративное решение, которое позволяет управлять процессами найма, оценивать кандидатов и создавать базы данных. Продукт хорошо масштабируется, но его высокая стоимость делает его менее доступным для небольших агентств.

1. Специализированные решения для рекрутинга: к этой категории относятся системы, ориентированные именно на автоматизацию процессов подбора персонала:

* HuntFlow

Популярная система среди кадровых агентств, она позволяет создавать базы данных кандидатов, вести карточки со всей необходимой информацией и автоматизировать этапы подбора. Преимуществом является удобный интерфейс и интеграция с сайтами для размещения вакансий. Основной недостаток — сложность кастомизации для уникальных процессов агентства.

* TalentLyft

Этот инструмент ориентирован на подбор персонала через социальные сети и аналитические платформы. Он хорошо подходит для агентств, которые активно используют маркетинговые методы в рекрутинге. Однако отсутствие поддержки русского языка может ограничить его использование на отечественном рынке.

1. Платформы для размещения вакансий и работы с кандидатами:

* HeadHunter (HH.ru)

Этот портал предоставляет агентствам доступ к обширной базе резюме и вакансий. Однако он не является полноценной системой автоматизации, а лишь предоставляет инструменты для поиска кандидатов.

* Superjob

Подобно HeadHunter, Superjob предлагает инструменты для работы с базами данных, но также не обеспечивает функционал для управления процессами подбора.

1. CRM-системы для кадровых агентств: некоторые кадровые агентства используют стандартные CRM-системы, такие как Bitrix24, адаптированные под их нужды. Такие решения позволяют:

* Управлять взаимоотношениями с клиентами.
* Вести учет вакансий и кандидатов.
* Автоматизировать бизнес-процессы, связанные с рекрутингом.

Преимуществом CRM-систем является их универсальность и возможность настройки. Однако они изначально не ориентированы на рекрутинг, что может потребовать значительных усилий для их адаптации.

1. Самописные системы: также некоторые агентства разрабатывают собственные программные решения. Это позволяет учитывать все специфические требования бизнеса, такие как:

* Учет уникальных показателей оценки кандидатов.
* Интеграция с внутренними системами компании.
* Полная кастомизация интерфейса.

Недостатком такого подхода являются высокие затраты на разработку и поддержку системы.

## Постановка задачи. Структура входной и выходной информации

Для реализации задачи требуется создание приложения, предоставляющего функции работы с данными о кандидатах, вакансиях и работодателях. Эти данные должны быть представлены в структурированном виде для упрощения взаимодействия пользователя с системой.

Цель разработки — автоматизация процессов подбора персонала с использованием цифрового приложения. Оно должно выполнять следующие функции:

1. Обеспечивать возможность добавления, редактирования и удаления записей о кандидатах, вакансиях и работодателях.
2. Предоставлять пользователю инструменты поиска кандидатов по заданным навыкам и вакансий по их требованиям.
3. Обеспечивать сохранение данных в локальной базе данных и их удобное представление в пользовательском интерфейсе.

Структура входной информации:

* Информация о кандидатах: имя, список навыков, количество лет опыта работы. Эти данные могут быть введены пользователем вручную через графический интерфейс или загружены из предварительно подготовленного файла в формате JSON.
* Данные о вакансиях: название вакансии, ключевые требования и идентификатор работодателя. Также поддерживается ввод через интерфейс или загрузка из файла.
* Информация о работодателях: название компании, отрасль и краткое описание. Аналогично, вводится вручную либо загружается.

Структура выходной информации:

* Визуальное представление данных в таблицах для кандидатов, вакансий и работодателей. Каждая таблица отображает основные параметры объектов и позволяет проводить операции с данными (например, удаление или редактирование).
* Результаты выполнения поиска: например, список подходящих кандидатов по конкретным навыкам или вакансии, которые соответствуют критериям работодателя. Результаты отображаются в отдельном окне или секции интерфейса.

Входные данные задают начальную базу для работы приложения, а выходные данные обеспечивают удобное и наглядное представление обработанной информации для пользователя. Такое решение позволяет кадровым агентствам повысить эффективность подбора персонала за счет автоматизации рутинных процессов и упрощения поиска информации.

# ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА СИСТЕМЫ КЛАССОВ

## Построение модели системы

Разработка системы классов основывается на ключевых объектах предметной области. Основные классы:

1. Candidate (Кандидат):

* Содержит информацию о кандидате: имя, навыки, опыт работы.
* Обеспечивает хранение и доступ к данным, а также возможность редактирования.

1. Vacancy (Вакансия):

* Представляет описание вакансии, включающее название, идентификатор работодателя, требования.
* Обеспечивает хранение данных и возможность их изменения.

1. HRDatabase (База данных):

* Управляет хранением и обработкой данных о кандидатах, вакансиях и работодателях.
* Предоставляет методы добавления, редактирования, удаления и поиска данных.

1. HRApp (Приложение):

* Основной класс, отвечающий за графический интерфейс и взаимодействие с пользователем.
* Использует PyQt5 для создания интерфейса.

Связь между классами реализована через базу данных, обеспечивающую обмен данными и сохранение их состояния.

## Описание среды разработки

При выборе инструментов для реализации моего проекта «Кадровое агентство» был проведен анализ доступных технологий для разработки приложений. Основной выбор пал на язык программирования Python и графическую библиотеку PyQt5.

Python был выбран в качестве основного языка разработки благодаря следующим преимуществам:

* Простота и читаемость кода: Python имеет простой синтаксис, что ускоряет процесс разработки и облегчает внесение изменений.
* Богатая экосистема: Язык предлагает множество библиотек для работы с базами данных (sqlite3), тестирования (pytest), и создания пользовательских интерфейсов (PyQt5).
* Независимость от платформ: Python-код можно запускать на различных операционных системах.

Сравнение Python с языком С представлено в таблице 1.

Таблица 1 - Сравнение языков программирования

|  |  |  |
| --- | --- | --- |
| Критерии | Python | C |
| Скорость обработки | 3 | 5 |
| Простота | 5 | 3 |
| Понятный синтаксис | 5 | 3 |
| Универсальность языка | 5 | 4 |
| Библиотеки | 5 | 3 |
| Итого: | 23 | 18 |

Так как Python набрал наибольшее количество баллов, это делает его идеальным выбором для создания систем автоматизации.

Для разработки графического интерфейса выбрана библиотека PyQt5. Ее преимущества:

* Возможность создания сложных многоуровневых интерфейсов.
* Наличие визуального редактора для ускорения разработки.
* Простая интеграция с Python-кодом и базой данных.

В качестве среды разработки используется PyCharm, предоставляющий:

* Авто дополнение кода и встроенный дебаггер.
* Управление виртуальными окружениями для обеспечения независимости проекта от системных библиотек.

Для управления базой данных использовался встроенный модуль sqlite3. Это решение было выбрано благодаря его простоте, надежности и отсутствию необходимости в дополнительной установке серверного ПО.

## Построение диаграммы прецедентов и диаграммы классов

Диаграмма прецедентов отображает действия пользователя в системе, где пользователем является менеджер кадрового агентства, взаимодействующий с базой данных через графический интерфейс. Основные действия включают:

* Добавление данных: ввод информации о новых кандидатах, вакансиях для актуализации базы данных.
* Поиск кандидатов и вакансий: фильтрация по ключевым параметрам для подбора подходящих кандидатов и вакансий.
* Редактирование данных: внесение изменений в резюме, требования вакансий или закрытие вакансий.

Диаграмма прецедентов приведена на рисунке 1.

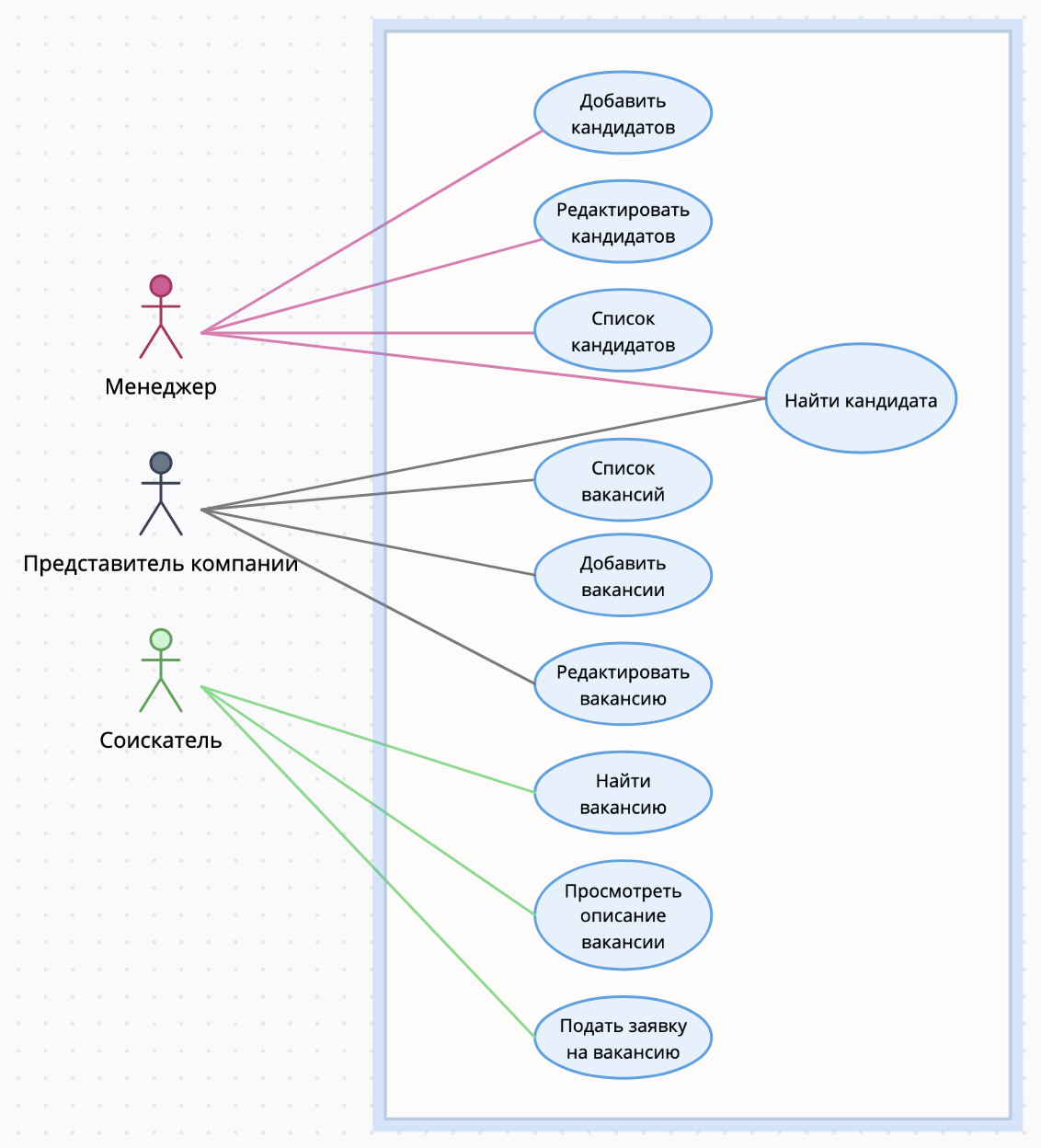


Рисунок 1 - Диаграмма прецедентов

После анализа взаимодействий проектируется архитектура системы, основанная на сущностях работодателей, кандидатов и вакансий, которые формируют основу для классов.

* Класс Candidate: включает данные о кандидатах — имя, навыки, опыт работы. Это упрощает поиск и редактирование информации, например, кандидат “Иван Иванов” с навыками веб-разработки и опытом 3 года.
* Класс Vacancy: хранит информацию о вакансиях, включая должность, идентификатор работодателя и требования к кандидатам. Например, вакансия “Разработчик Python” от “TechCorp”.

Диаграмма классов приведена на рисунке 2.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 2 - Диаграмма классов

## Описание модулей

Система «Кадровое агентство» реализована с использованием модульной архитектуры. Основные модули отвечают за управление данными, взаимодействие с пользователем через графический интерфейс и бизнес-логику.

Главный модуль (*main.py*):

Главный модуль проекта представляет собой основное приложение для работы с кадровым агентством, включающее удобный графический интерфейс. Он создан с использованием библиотеки PyQt5 и обеспечивает взаимодействие между пользователем и базой данных. Основной задачей модуля является управление данными о кандидатах и вакансиях:

1. Работа с кандидатами:

* Добавление новых кандидатов с указанием имени, навыков и опыта работы.
* Редактирование данных уже существующих кандидатов.
* Просмотр списка всех кандидатов в удобной табличной форме.

1. Работа с вакансиями:

* Добавление новых вакансий с указанием описания, требований и работодателя.
* Редактирование информации о существующих вакансиях.
* Просмотр списка вакансий с возможностью анализа данных.

1. Поиск и сопоставление:

* Поиск подходящих кандидатов под конкретные вакансии по навыкам.
* Поиск вакансий, соответствующих навыкам кандидата.

Приложение автоматически инициализирует базу данных при первом запуске, добавляя в нее тестовые данные для демонстрации работы. Это позволяет сразу протестировать функции и оценить работу приложения.

Главный модуль содержит основной класс HRApp, который отвечает за создание интерфейса, обработку событий и вызовы методов базы данных. При запуске приложения отображается главное окно с приветствием и кнопками для доступа к основным функциям. Пользователь может выполнять необходимые операции через всплывающие диалоговые окна и таблицы.

Полный код этого модуля представлен в *Приложении 2 (Листинг 1).*

Модуль с классами (*classes.py*):

Этот модуль содержит основные классы, используемые для работы с данными кадрового агентства. Классы описывают структуру данных, необходимые для обработки информации о кандидатах и вакансиях.

1. Класс Candidate:

* name — имя кандидата.
* skills — перечень профессиональных навыков кандидата.
* experience — количество лет профессионального опыта.

Класс используется для хранения и обработки данных о соискателях, таких как их навыки и стаж, что позволяет находить подходящие вакансии.

1. Класс Vacancy:

* title — название вакансии.
* employer\_id — идентификатор работодателя, связанного с вакансией.
* requirements — требования к кандидатам, указанные в вакансии.

Класс используется для хранения данных о вакансиях, включая их описание и требования, что упрощает процесс сопоставления с подходящими кандидатами.

Эти классы являются основой для работы с данными приложения, обеспечивая их четкую структуру и упрощая взаимодействие с базой данных.

Полный код этого модуля представлен в *Приложении 2 (Листинг 2).*

Модуль для работы с базой данных (hr\_database.py):

Модуль предназначен для работы с базой данных в рамках проекта кадрового агентства. Он отвечает за управление информацией о кандидатах и вакансиях, обеспечивая удобный доступ к данным и их обработку. Данные хранятся в локальной базе SQLite, что упрощает развертывание и использование системы:

1. Создание базы данных и таблиц: при запуске автоматически создаются таблицы для хранения данных о кандидатах и вакансиях, если они еще не существуют.
2. Добавление новых записей: модуль позволяет добавлять информацию о кандидатах и вакансиях, избегая дублирования данных.
3. Редактирование записей: реализована функция обновления данных кандидатов, включая имя, навыки и опыт работы.
4. Получение данных: возможен вывод всех записей или выборка по заданным критериям. Например, можно получить список кандидатов с конкретными навыками или все доступные вакансии.
5. Поиск соответствий: поддерживается поиск кандидатов, подходящих под требования вакансий, а также подбор вакансий, соответствующих навыкам кандидатов.

Этот модуль является ключевой частью системы, связывая функциональность базы данных с другими компонентами проекта.

Полный код этого модуля представлен в *Приложении 2 (Листинг 3).*

## Расчет сложности алгоритма

Анализ сложности алгоритмов в приложении позволяет оценить их производительность и устойчивость к увеличению объема данных. Рассмотрим основные операции, выполняемые в разработанном приложении, и вычислим их временную сложность, используя нотацию Big O.

1. Добавление записей (кандидат, вакансия, работодатель):

Описание: при добавлении новых записей приложение формирует SQL-запрос вида `INSERT INTO`, который отправляется в базу данных SQLite.

Сложность: О(1) - операция вставки является постоянной по времени, поскольку система обращается к индексу и добавляет данные без перебора существующих записей.

1. Редактирование записей:

Описание: редактирование осуществляется с помощью SQL-запроса `UPDATE`, обновляющего одну или несколько строк в базе данных на основании условий.

Сложность: O(1) - операция обновления выполняется мгновенно, так как система непосредственно обновляет данные в указанной строке.

1. Поиск по навыкам:

Описание: для поиска кандидатов по навыкам приложение перебирает записи базы данных и проверяет соответствие списка навыков кандидатов заданным критериям.

Сложность: O(*n*), где *n* — количество записей в таблице. Это линейная сложность, поскольку каждую запись необходимо обработать. Однако при добавлении индекса для поля навыков сложность может быть снижена.

1. Отображение данных:

Описание: вывод данных из базы в таблицу требует извлечения строк и их последующего представления в интерфейсе.

Сложность: O(*n\*m*), где *n* — количество строк, *m* — количество столбцов. Каждая ячейка таблицы обрабатывается отдельно для формирования итогового вида.

1. Фильтрация данных (вакансии, кандидаты):

Описание: фильтрация заключается в применении условий поиска (например, стаж, количество навыков) к базе данных.

Сложность: O(*n*) - все записи просматриваются, чтобы найти те, которые соответствуют заданным критериям. В случае использования индексов сложность может быть снижена до O(*log n*).

1. Удаление записей:

Описание: удаление записей осуществляется через SQL-запрос `DELETE`, который удаляет строки по указанному условию.

Сложность: O(1) - операция удаления также не зависит от количества записей, поскольку выполняется над строго определенными строками.

Этот анализ демонстрирует, что текущая реализация приложения удовлетворяет требованиям производительности для типичных объемов данных кадрового агентства.

# ТЕСТИРОВАНИЕ И ОТЛАДКА СИСТЕМЫ

## Тестирование системы методом черного ящика

Метод тестирования «черный ящик» используется для проверки функциональности системы без анализа ее внутреннего устройства. Цель такого тестирования — убедиться, что система выполняет свои функции в соответствии с требованиями.

Тест 1: Добавление кандидата: пользователь вводит корректные данные о кандидате через интерфейс.

Действия:

1. Открыть окно добавления кандидата.
2. Ввести данные:

* Имя: Иван Иванов.
* Навыки: Python, SQL.
* Опыт: 5 лет.

1. Нажать кнопку «Добавить».

Ожидаемый результат: кандидат добавляется в базу данных, и в таблице отображается новая запись.

Тест 2: Добавление кандидата с некорректными данными: пользователь оставляет одно из полей пустым или вводит некорректный формат данных.

Действия:

1. Открыть окно добавления кандидата.
2. Ввести данные:

* Имя: пустое поле.
* Навыки: Java, Spring.
* Опыт: строка вместо числа.

1. Нажать кнопку «Добавить».

Ожидаемый результат: появляется сообщение об ошибке с запросом корректного ввода.

Тест 3: Поиск кандидатов: пользователь ищет кандидатов по навыку «Python».

Действия:

1. Открыть окно поиска.
2. Ввести навык: Python.
3. Нажать кнопку «Искать».

Ожидаемый результат: отображаются только те кандидаты, которые имеют навык Python.

Тест 4: Редактирование данных: пользователь изменяет информацию о существующем кандидате.

Действия:

1. Открыть список кандидатов.
2. Выбрать запись для редактирования.
3. Изменить данные:

* Имя: Анна Смирнова.
* Опыт: 7 лет.

1. Нажать кнопку «Сохранить изменения».

Ожидаемый результат: данные обновляются в базе и таблице.

Тест 5: Обработка пустой базы данных: пользователь открывает список кандидатов, когда база данных пуста.

Действия:

1. Открыть окно просмотра кандидатов.

Ожидаемый результат: появляется сообщение: «Данные отсутствуют».

План тестирования представлен в таблице 2.

Таблица 2 - План тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID теста | Описание | Входные данные | Ожидаемый результат | Статус |
| TC-001 | Добавление кандидата | Иван Иванов, Python, 5 | Кандидат добавлен в базу | Успешно |
|  | | | | |
| TC-002 | Добавление некорректных данных | Пустое поле, Java, текст | Ошибка: «Введите корректные данные!» | Успешно |
|  | | | | |
| TC-003 | Поиск кандидатов | Python | Отображаются кандидаты с навыком Python | Успешно |
|  | | | | |
| TC-004 | Редактирование данных | Обновленные данные | Данные обновлены в базе и таблице | Успешно |
| Изображение выглядит как текст, снимок экрана, мультимедиа, гаджет  Автоматически созданное описание | | | | |
| TC-005 | Отображение пустой базы | Нет записей | Пустая база данных | Успешно |
| Изображение выглядит как снимок экрана, текст, мультимедиа, программное обеспечение  Автоматически созданное описание | | | | |

## Описание применения средств отладки

В данном разделе демонстрируется пример отладки, которая была выполнена в процессе разработки кода.

Во время разработки приложения возникла ошибка при загрузке начальных данных из файла input\_data.json (*Рисунок 3*). Сообщение об ошибке, выведенное в консоль, выглядело следующим образом:

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

Автоматически созданное описаниеОшибка загрузки данных: [Errno 2] No such file or directory: 'input\_data.json'

Рисунок 3. Скриншот ошибки

Данная ошибка указывает на то, что программа не может найти файл input\_data.json в указанной директории. Причиной проблемы оказалось отсутствие этого файла в папке проекта.

Шаги по устранению ошибки

* + 1. Создание тестового файла с начальными данными:

Был создан файл input\_data.json, содержащий тестовые данные в формате JSON. Структура файла была составлена в соответствии с предполагаемыми объектами системы:

* + Кандидаты (Candidates) с именем, навыками и опытом работы.
  + Вакансии (Vacancies) с названием, ID работодателя и требованиями.
    1. Размещение файла в директории проекта:

Файл input\_data.json был помещён в ту же директорию, где находится главный модуль main.py.

* + 1. Повторный запуск программы:

После добавления файла ошибка при загрузке данных исчезла, и начальные данные успешно загрузились в базу данных.

Проблема была успешно решена путём анализа ошибки и добавления необходимого файла с корректными начальными данными. После устранения ошибки программа продолжила работать корректно, а данные отобразились в пользовательском интерфейсе.

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была разработана современная система «Кадровое агентство», автоматизирующая ключевые HR-процессы: управление данными о вакансиях и кандидатах. Основной модуль включает удобный графический интерфейс с функциями добавления, редактирования, поиска и отображения информации, что делает приложение универсальным инструментом для специалистов по подбору персонала. Связь интерфейса с базой данных реализована через специально разработанный класс, обеспечивающий надёжное хранение данных и высокую производительность за счёт оптимизированных SQL-запросов и эффективной структуры таблиц. Для загрузки входных данных использовался формат JSON.

В будущем планируется расширение системы. В частности, предлагается внедрить модуль авторизации, который обеспечит разграничение прав доступа пользователей, повысив безопасность и гибкость. Также планируется реализовать функции построения аналитических отчетов с визуализацией ключевых метрик и возможность экспорта данных в форматах CSV и PDF, что сделает систему ещё более удобной для использования (*Приложение 3*).

Оптимизация работы с базой данных, позволит повысить производительность при работе с большими объёмами информации. Развитие интерфейса с применением интерактивных элементов, таких как текстовые поля, кнопки, выпадающие списки и чек-боксы, обеспечит более интуитивное взаимодействие с системой.

Созданная система уже является эффективным инструментом для автоматизации HR-процессов. Реализация запланированных улучшений усилит её функционал, сделав более востребованной и мощной. Проект демонстрирует важность комплексного подхода к проектированию, разработке и тестированию программных решений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Стандарты и законодательные материалы

1. Федеральный закон от 27 июля 2006 г. N 149-ФЗ "Об информации, информационных технологиях и о защите информации".
2. Федеральный закон от 26 июля 2017 г. N 187-ФЗ "О безопасности критической информационной структуры Российской Федерации".
3. ГОСТ 7.32–2001 «Отчет о научно-исследовательской работе. Структура и правила оформления»
4. ГОСТ 7.80–2000 «Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления»
5. ГОСТ 19.701–90 (ИСО 5807–85) — «Единая система программной документации. Схемы алгоритмов, программ, данных и систем».
6. ГОСТ 9327–60. Форматы бумажных изданий. Потребительские форматы.

Учебники и учебные пособия

1. Любанович, Б. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.
2. Свейгарт, Э. Автоматизация рутинных задач с помощью Python. —2016. — 582 с.
3. Молинаро, Э., де Грааф, Р. SQL. Сборник рецептов. — 2009. — 668 с.
4. Ли, К. “A Practitioner's Guide to Software Test Design”. [Электронный ресурс]: Rulit.me – Режим доступа: <https://www.rulit.me/tag/other-computers/a-practitioner-s-guide-to-software-test-design-perevod-download-668733.html>
5. Бассет, Л. Введение в JSON (Introduction to JavaScript Object Notation, L. Bassett). — 2015.

Интернет-источники

1. Руководство по SQLite // <https://metanit.com/sql/sqlite>
2. Балакирев, С. В. Добрый, добрый Python с Сергеем Балакиревым // <https://stepik.org/course/100707/syllabus>
3. Объектно-ориентированное программирование. Классы и объекты // <https://metanit.com/python/tutorial/7.1.php>
4. Оправка данных в формате JSON в Python // <https://metanit.com/python/django/3.9.php>
5. PyQt5 Signals, Slots & Events (Сигналы, слоты и события PyQt5) // <https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/>
6. Python.org. "Документация по языку программирования Python" // <https://docs.python.org>.
7. Кадровое агентство HR-PROFI-№1 по подбору персонала // [https://hr-profi.ru](https://hr-profi.ru/)
8. JetBrains The Python IDE for data science and web development // <https://www.jetbrains.com/pycharm/>
9. Работа с файлами в формате JSON // <https://pyneng.readthedocs.io/ru/latest/book/17_serialization/json.html>
10. Сложность алгоритмов. Разбор Big O. Open-JS // <https://habr.com/ru/articles/782608/>

# ПРИЛОЖЕНИЕ

## Приложение 1. Блок-схемы

*Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание*

Рисунок 4 - Блок-схема главного модуля

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 5 - Блок-схема модуля «добавление кандидата»

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рисунок 6 - Блок-схема модуля «просмотр данных»

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 7 - Блок-схема модуля «добавление вакансии»

Изображение выглядит как диаграмма, линия, текст, План

Автоматически созданное описание

Рисунок 8 - Блок-схема модуля «поиск кандидатов/вакансий»

## Приложение 2. Код программы

Листинг 1.

*"""Главный модуль"""*

import json  
import sys  
from PyQt5.QtWidgets import (QApplication, QMainWindow, QVBoxLayout, QHBoxLayout, QWidget, QLabel, QLineEdit,  
 QPushButton, QTableWidget, QTableWidgetItem, QMessageBox, QDialog, QInputDialog)  
from PyQt5.QtCore import Qt  
from classes import Candidate, Vacancy  
from hr\_database import HRDatabase  
  
  
  
# Главный класс приложения  
class HRApp(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Кадровое агентство")  
 self.db = HRDatabase()  
  
 # Загрузка начальных данных из файла  
 try:  
 self.db.load\_from\_file("input\_data.json") # Укажите путь к вашему файлу  
 except Exception as e:  
 print(f"Ошибка загрузки данных: {e}")  
  
 self.init\_ui()  
 def init\_ui(self):  
 main\_widget = QWidget()  
 self.setCentralWidget(main\_widget)  
 main\_layout = QVBoxLayout()  
  
 label = QLabel('Добро пожаловать в кадровое агентство "SELA"!') # Приветственное сообщение  
 label.setAlignment(Qt.AlignCenter)  
  
 main\_layout.addWidget(label)  
 main\_widget.setLayout(main\_layout)  
  
 # Основной горизонтальный макет для разделения блоков  
 main\_horizontal\_layout = QHBoxLayout()  
  
 # Блок для кандидатов  
 candidates\_layout = QVBoxLayout()  
 candidates\_label = QLabel("Кандидаты")  
 candidates\_label.setAlignment(Qt.AlignCenter)  
 add\_candidate\_btn = QPushButton("Добавить кандидата")  
 add\_candidate\_btn.clicked.connect(self.show\_add\_candidate\_dialog)  
 edit\_candidate\_btn = QPushButton("Редактировать кандидата")  
 edit\_candidate\_btn.clicked.connect(self.show\_edit\_candidate\_dialog)  
 view\_candidates\_btn = QPushButton("Список кандидатов")  
 view\_candidates\_btn.clicked.connect(self.show\_candidates)  
  
 candidates\_layout.addWidget(candidates\_label)  
 candidates\_layout.addWidget(add\_candidate\_btn)  
 candidates\_layout.addWidget(edit\_candidate\_btn)  
 candidates\_layout.addWidget(view\_candidates\_btn)  
  
 # Блок для вакансий  
 vacancies\_layout = QVBoxLayout()  
 vacancies\_label = QLabel("Вакансии")  
 vacancies\_label.setAlignment(Qt.AlignCenter)  
 add\_vacancy\_btn = QPushButton("Добавить вакансию")  
 add\_vacancy\_btn.clicked.connect(self.show\_add\_vacancy\_dialog)  
 edit\_vacancy\_btn = QPushButton("Редактировать вакансию")  
 edit\_vacancy\_btn.clicked.connect(self.show\_edit\_vacancy\_dialog)  
 view\_vacancies\_btn = QPushButton("Список вакансий")  
 view\_vacancies\_btn.clicked.connect(self.show\_vacancies)  
  
 vacancies\_layout.addWidget(vacancies\_label)  
 vacancies\_layout.addWidget(add\_vacancy\_btn)  
 vacancies\_layout.addWidget(edit\_vacancy\_btn)  
 vacancies\_layout.addWidget(view\_vacancies\_btn)  
  
 # Кнопка для поиска вакансий и кандидатов  
 match\_vacancies\_btn = QPushButton("Поиск вакансий и кандидатов")  
 match\_vacancies\_btn.clicked.connect(self.show\_match\_candidates)  
  
 # Добавление столбцов и кнопок в основной макет  
 main\_horizontal\_layout.addLayout(candidates\_layout)  
 main\_horizontal\_layout.addLayout(vacancies\_layout)  
  
 main\_layout.addLayout(main\_horizontal\_layout)  
 main\_layout.addWidget(match\_vacancies\_btn)  
  
 main\_widget.setLayout(main\_layout)  
  
 def show\_edit\_vacancy\_dialog(self):  
 # Диалоговое окно для редактирования вакансий  
 vacancies = self.db.get\_vacancies()  
 vacancy\_ids = [str(v[0]) for v in vacancies]  
 selected\_id, ok = QInputDialog.getItem(self, "Редактировать вакансию", "Выберите ID вакансии:", vacancy\_ids, 0,  
 False)  
  
 if ok:  
 dialog = QDialog(self)  
 dialog.setWindowTitle("Редактировать вакансию")  
 layout = QVBoxLayout()  
 vacancy = next(v for v in vacancies if v[0] == int(selected\_id))  
  
 title\_input = QLineEdit(vacancy[1]) # Название вакансии  
 employer\_input = QLineEdit(str(vacancy[2])) # ID работодателя  
 requirements\_input = QLineEdit(vacancy[3]) # Требования к вакансии  
  
 def update\_vacancy():  
 # Обновление данных вакансии  
 title = title\_input.text().strip()  
 employer\_id = employer\_input.text().strip()  
 requirements = requirements\_input.text().strip()  
 if not title or not employer\_id.isdigit() or not requirements:  
 QMessageBox.warning(dialog, "Ошибка", "Введите корректные данные!")  
 return  
 self.db.conn.execute('''UPDATE Vacancies   
 SET title = ?, employer\_id = ?, requirements = ?   
 WHERE id = ?''',  
 (title, int(employer\_id), requirements, int(selected\_id)))  
 QMessageBox.information(dialog, "Успех", "Вакансия обновлена!")  
 dialog.accept()  
  
 submit\_btn = QPushButton("Сохранить изменения")  
 submit\_btn.clicked.connect(update\_vacancy)  
  
 layout.addWidget(title\_input)  
 layout.addWidget(employer\_input)  
 layout.addWidget(requirements\_input)  
 layout.addWidget(submit\_btn)  
 dialog.setLayout(layout)  
 dialog.exec\_()  
  
 def show\_add\_vacancy\_dialog(self):  
 # Создаем диалоговое окно для добавления вакансии  
 dialog = QDialog(self)  
 dialog.setWindowTitle("Добавить вакансию")  
 layout = QVBoxLayout()  
  
 # Создаем поля ввода для описания вакансии, ID работодателя и требований  
 title\_input = QLineEdit()  
 title\_input.setPlaceholderText("Описание (через запятую)")  
 employer\_input = QLineEdit()  
 employer\_input.setPlaceholderText("ID работодателя")  
 requirements\_input = QLineEdit()  
 requirements\_input.setPlaceholderText("Требования")  
  
 # Функция для добавления вакансии в базу данных  
 def add\_vacancy():  
 title = title\_input.text().strip()  
 employer\_id = employer\_input.text().strip()  
 requirements = requirements\_input.text().strip()  
  
 # Проверка корректности введенных данных  
 if not title or not employer\_id.isdigit() or not requirements:  
 QMessageBox.warning(dialog, "Ошибка", "Введите корректные данные!")  
 return  
  
 # Создаем объект вакансии и добавляем его в базу данных  
 vacancy = Vacancy(title, int(employer\_id), requirements)  
 self.db.add\_vacancy(vacancy)  
  
 # Показываем сообщение об успешном добавлении вакансии  
 QMessageBox.information(dialog, "Успех", "Вакансия добавлена!")  
 dialog.accept()  
  
 # Создаем кнопку для добавления вакансии  
 submit\_btn = QPushButton("Добавить")  
 submit\_btn.clicked.connect(add\_vacancy)  
  
 layout.addWidget(title\_input)  
 layout.addWidget(employer\_input)  
 layout.addWidget(requirements\_input)  
 layout.addWidget(submit\_btn)  
 dialog.setLayout(layout)  
 dialog.exec\_()  
  
 def show\_add\_candidate\_dialog(self):  
 # Создаем диалоговое окно для добавления нового кандидата  
 dialog = QDialog(self)  
 dialog.setWindowTitle("Добавить кандидата") # заголовок окна  
 layout = QVBoxLayout()  
 # Создаем поля ввода для имени, навыков и опыта кандидата  
 name\_input = QLineEdit()  
 name\_input.setPlaceholderText("Имя (Имя и Фамилия)")  
 skills\_input = QLineEdit()  
 skills\_input.setPlaceholderText("Навыки")  
 experience\_input = QLineEdit()  
 experience\_input.setPlaceholderText("Опыт (в годах)")  
  
 # Функция для добавления нового кандидата в базу данных  
 def add\_candidate():  
 name = name\_input.text().strip()  
 skills = skills\_input.text().strip()  
 experience = experience\_input.text().strip()  
 # Проверка корректности введенных данных  
 if not name or not skills or not experience.isdigit():  
 QMessageBox.warning(dialog, "Ошибка", "Введите корректные данные!")  
 return  
 # Создаем объект кандидата и добавляем его в базу данных  
 candidate = Candidate(name, skills, int(experience))  
 self.db.add\_candidate(candidate)  
 # Показываем сообщение об успешном добавлении кандидата  
 QMessageBox.information(dialog, "Успех", "Кандидат добавлен!")  
 dialog.accept() # Закрываем диалог  
  
 # Создаем кнопку для добавления кандидата  
 submit\_btn = QPushButton("Добавить")  
 submit\_btn.clicked.connect(add\_candidate) # Привязываем действие кнопки к функции  
  
 layout.addWidget(name\_input)  
 layout.addWidget(skills\_input)  
 layout.addWidget(experience\_input)  
 layout.addWidget(submit\_btn)  
 dialog.setLayout(layout)  
 dialog.exec\_()  
  
 def show\_edit\_candidate\_dialog(self):  
 # Получаем список всех кандидатов из базы данных  
 candidates = self.db.get\_candidates()  
 candidate\_ids = [str(c[0]) for c in candidates] # Извлекаем ID кандидатов для выбора  
  
 selected\_id, ok = QInputDialog.getItem(self, "Редактировать кандидата", "Выберите ID кандидата:", candidate\_ids,  
 0, False)  
  
 if ok:  
 # диалоговое окно для редактирования данных выбранного кандидата  
 dialog = QDialog(self)  
 dialog.setWindowTitle("Редактировать кандидата")  
 layout = QVBoxLayout()  
  
 # Заполняем поля ввода текущими данными кандидата  
 name\_input = QLineEdit(candidates[int(selected\_id) - 1][1])  
 skills\_input = QLineEdit(candidates[int(selected\_id) - 1][2])  
 experience\_input = QLineEdit(str(candidates[int(selected\_id) - 1][3]))  
  
 # Функция для обновления данных кандидата в базе данных  
 def update\_candidate():  
 name = name\_input.text().strip()  
 skills = skills\_input.text().strip()  
 experience = experience\_input.text().strip()  
 # Проверка корректности введенных данных  
 if not name or not skills or not experience.isdigit():  
 QMessageBox.warning(dialog, "Ошибка", "Введите корректные данные!")  
 return  
 # Обновляем данные кандидата в базе данных  
 self.db.edit\_candidate(int(selected\_id), name, skills, int(experience))  
 # Показываем сообщение об успешном обновлении данных  
 QMessageBox.information(dialog, "Успех", "Данные кандидата обновлены!")  
 dialog.accept()  
  
 # Создаем кнопку для сохранения изменений  
 submit\_btn = QPushButton("Сохранить")  
 submit\_btn.clicked.connect(update\_candidate) # Привязываем действие кнопки к функции  
  
 layout.addWidget(name\_input)  
 layout.addWidget(skills\_input)  
 layout.addWidget(experience\_input)  
 layout.addWidget(submit\_btn)  
 dialog.setLayout(layout)  
 dialog.exec\_()  
  
 def show\_candidates(self):  
 # Отображаем таблицу с кандидатами, получая данные из базы  
 self.show\_table("Кандидаты", self.db.get\_candidates())  
  
 def show\_vacancies(self):  
 # Отображаем таблицу с вакансиями, получая данные из базы  
 self.show\_table("Вакансии", self.db.get\_vacancies())  
  
 def show\_match\_candidates(self):  
 # Диалог для выбора категории поиска (вакансии или кандидаты)  
 category, ok = QInputDialog.getItem(  
 self,  
 "Поиск",  
 "Выберите категорию для поиска:",  
 ["Поиск вакансий", "Поиск кандидатов"],  
 0,  
 False  
 )  
 if ok and category:  
 # Диалог для ввода навыка, по которому будет производиться поиск  
 skill, ok = QInputDialog.getText(self, f"Поиск для {category}", "Введите навык (например, Python):")  
 if ok and skill.strip():  
 skill = skill.strip() # лишние пробелы  
 # В зависимости от выбранной категории поиска, выполняем поиск по навыку  
 if category == "Поиск вакансий":  
 results = self.db.find\_vacancies\_by\_skill(skill) # Ищем вакансии по навыку  
 self.show\_table(f"Вакансии для навыка: {skill}", results)  
 elif category == "Поиск кандидатов":  
 results = self.db.find\_matching\_candidates(skill) # Ищем кандидатов по навыку  
 self.show\_table(f"Кандидаты для навыка: {skill}", results)  
  
 def show\_table(self, title, data):  
 # Отображаем таблицу с данными (например, кандидаты или вакансии)  
 dialog = QDialog(self)  
 dialog.setWindowTitle(title)  
 layout = QVBoxLayout()  
  
 # Создаем таблицу для отображения данных  
 table = QTableWidget()  
 if data:  
 # Если данные есть, устанавливаем количество строк и столбцов  
 table.setRowCount(len(data))  
 table.setColumnCount(len(data[0]))  
 for row, record in enumerate(data):  
 for col, item in enumerate(record):  
 table.setItem(row, col, QTableWidgetItem(str(item)))  
  
 layout.addWidget(table)  
 dialog.setLayout(layout)  
 dialog.exec\_()  
  
 # заливаем данные в базу только если таблицы пусты  
 db = HRDatabase() # создаём объект для работы с базой данных  
  
 # проверяем, есть ли кандидаты в базе  
 existing\_candidates = db.get\_candidates()  
 if not existing\_candidates:  
 # если кандидаты не найдены, добавляем их  
 for candidate in db.get\_candidates():  
 db.add\_candidate(candidate) # добавляем кандидата в базу  
  
 # проверяем, есть ли вакансии в базе  
 existing\_vacancies = db.get\_vacancies()  
 if not existing\_vacancies:  
 # если вакансии не найдены, добавляем их  
 for idx, vacancy in enumerate(db.get\_vacancies(), start=1):  
 vacancy.employer\_id = idx # связываем вакансии с работодателями (например, уникальные ID)  
 db.add\_vacancy(vacancy) # добавляем вакансию в базу  
  
# основной блок для запуска приложения  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = HRApp()  
 window.show()  
 sys.exit(app.exec\_())

Листинг 2.

*"""Модуль, где инициализируются классы"""*# Модели данных для кандидатов и вакансий  
class Candidate:  
 def \_\_init\_\_(self, name, skills, experience):  
 self.name = name # Имя кандидата  
 self.skills = skills # Навыки кандидата  
 self.experience = experience # Опыт кандидата (в годах)  
  
class Vacancy:  
 def \_\_init\_\_(self, title, employer\_id, requirements):  
 self.title = title # Название вакансии  
 self.employer\_id = employer\_id # ID работодателя, которому принадлежит вакансия  
 self.requirements = requirements # Требования к вакансии

Листинг 3.

*"""Модуль для работы с базой данных"""*import json  
import sqlite3  
from classes import Candidate, Vacancy  
  
# Класс для работы с базой данных  
class HRDatabase:  
 def \_\_init\_\_(self, db\_name="кадровое агентство сельгира.db"):  
 self.conn = sqlite3.connect(db\_name) # Подключение к базе данных  
 self.create\_tables() # Создание таблиц при инициализации базы данных  
  
 def create\_tables(self):  
 # Создание таблиц в базе данных (если их нет)  
 with self.conn:  
 self.conn.execute('''CREATE TABLE IF NOT EXISTS Candidates (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 skills TEXT NOT NULL,  
 experience INTEGER NOT NULL)''')  
 self.conn.execute('''CREATE TABLE IF NOT EXISTS Vacancies (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 title TEXT NOT NULL,  
 employer\_id INTEGER NOT NULL,  
 requirements TEXT,  
 FOREIGN KEY (employer\_id) REFERENCES Employers (id))''')  
  
 def add\_candidate(self, candidate: Candidate):  
 # Добавление нового кандидата в базу данных (если его нет)  
 with self.conn:  
 existing\_candidates = self.conn.execute("SELECT \* FROM Candidates WHERE name = ?",  
 (candidate.name,)).fetchall()  
 if not existing\_candidates: # Если кандидат еще не существует в базе  
 self.conn.execute("INSERT INTO Candidates (name, skills, experience) VALUES (?, ?, ?)",  
 (candidate.name, candidate.skills, candidate.experience))  
  
 def add\_vacancy(self, vacancy: Vacancy):  
 # Добавление новой вакансии в базу данных (если она не существует)  
 with self.conn:  
 existing\_vacancies = self.conn.execute("SELECT \* FROM Vacancies WHERE title = ?", (vacancy.title,)).fetchall()  
 if not existing\_vacancies: # Если вакансия еще не существует в базе  
 self.conn.execute("INSERT INTO Vacancies (title, employer\_id, requirements) VALUES (?, ?, ?)",  
 (vacancy.title, vacancy.employer\_id, vacancy.requirements))  
  
 def load\_from\_file(self, filename):  
 with open(filename, 'r', encoding='utf-8') as file:  
 if filename.endswith('.json'):  
 data = json.load(file)  
 elif filename.endswith('.txt'):  
 import ast  
 data = ast.literal\_eval(file.read())  
 else:  
 raise ValueError("Поддерживаются только файлы .json или .txt")  
  
 # Очистка таблиц перед загрузкой  
 with self.conn:  
 self.conn.execute("DELETE FROM Candidates")  
 self.conn.execute("DELETE FROM Vacancies")  
  
 # Загрузка данных в базу  
 candidates = data.get('candidates', [])  
 vacancies = data.get('vacancies', [])  
  
 for candidate in candidates:  
 self.add\_candidate(Candidate(candidate['name'], candidate['skills'], candidate['experience']))  
  
 for vacancy in vacancies:  
 self.add\_vacancy(Vacancy(vacancy['title'], vacancy['employer\_id'], vacancy['requirements']))  
  
 def get\_candidates(self, skill\_filter=None):  
 # Получение всех кандидатов (или кандидатов с определенными навыками)  
 with self.conn:  
 if skill\_filter:  
 return self.conn.execute("SELECT \* FROM Candidates WHERE skills LIKE ?", (f"%{skill\_filter}%",)).fetchall()  
 return self.conn.execute("SELECT \* FROM Candidates").fetchall()  
  
 def edit\_candidate(self, candidate\_id, name, skills, experience):  
 # Редактирование данных кандидата  
 with self.conn:  
 self.conn.execute('''UPDATE Candidates   
 SET name = ?, skills = ?, experience = ?   
 WHERE id = ?''', (name, skills, experience, candidate\_id))  
  
 def get\_vacancies(self):  
 # Получение всех вакансий  
 with self.conn:  
 return self.conn.execute("SELECT \* FROM Vacancies").fetchall()  
  
 def find\_matching\_candidates(self, requirements):  
 # Поиск кандидатов, соответствующих требованиям вакансии  
 with self.conn:  
 return self.conn.execute('''SELECT \* FROM Candidates WHERE skills LIKE ?''', (f"%{requirements}%",)).fetchall()  
  
 def add\_vacancy(self, vacancy: Vacancy):  
 with self.conn:  
 self.conn.execute("INSERT INTO Vacancies (title, employer\_id, requirements) VALUES (?, ?, ?)",  
 (vacancy.title, vacancy.employer\_id, vacancy.requirements))  
  
 def find\_vacancies\_by\_skill(self, skills):  
 # Поиск вакансий по навыкам  
 skill\_list = [skill.strip() for skill in skills.split(',')] # Разделение навыков по запятой  
 query = "SELECT \* FROM Vacancies WHERE " + " AND ".join(  
 ["requirements LIKE ?" for \_ in skill\_list])  
 params = [f"%{skill}%" for skill in skill\_list]  
 with self.conn:  
 return self.conn.execute(query, tuple(params)).fetchall()  
  
 def find\_matching\_candidates(self, skills):  
 # Поиск кандидатов по навыкам  
 skill\_list = [skill.strip() for skill in skills.split(',')] # Разделение навыков по запятой  
 query = "SELECT \* FROM Candidates WHERE " + " AND ".join(  
 ["skills LIKE ?" for \_ in skill\_list])  
 params = [f"%{skill}%" for skill in skill\_list]  
 with self.conn:  
 return self.conn.execute(query, tuple(params)).fetchall()

## Приложение 3. Сценарий работы программы

Изображение выглядит как текст, диаграмма, снимок экрана, План

Автоматически созданное описание

## Приложение 4. Графический интерфейс

Изображение выглядит как текст, человек, снимок экрана, одежда

Автоматически созданное описание

Рисунок 9 - Главная страница

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 10 - Вкладка "Поиск"