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**

Изображение выглядит как графическая вставка, Графика, мультфильм

Автоматически созданное описание

**УТВЕРЖДАЮ**

**Зам. директора КМПО**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С.Ф. Гасанов**

**«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.**

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

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

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

**Студент группы 302ИС-22 Сельгира Сангаджи-Горяева**

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

Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

Срок сдачи проекта «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

**Москва 2024**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc185035872)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc185035873)

[1.1. Описание предметной области и основные процессы кадрового агентства 5](#_Toc185035874)

[1.2. Важность автоматизации процессов кадрового агентства 6](#_Toc185035875)

[1.3. Задачи анализа данных и управления базой кадрового агентства 8](#_Toc185035876)

[2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА МОДУЛЯ 11](#_Toc185035877)

[2.1. Общее назначение системы 11](#_Toc185035878)

[2.2. Описание среды разработки 11](#_Toc185035879)

[2.3. Построение диаграммы прецедентов и диаграммы классов 12](#_Toc185035880)

[2.4. Выбор инструментов 14](#_Toc185035881)

[2.5. Описание модулей 16](#_Toc185035882)

[3. РЕАЛИЗАЦИЯ ПРОЕКТА СИСТЕМЫ 18](#_Toc185035883)

[3.1. Описание модулей 18](#_Toc185035884)

[3.1.1. Модуль моделей данных 18](#_Toc185035885)

[3.1.2. Модуль управления базой данных 18](#_Toc185035886)

[3.1.3. Модуль графического интерфейса 19](#_Toc185035887)

[3.2. Пример реализации главного модуля 19](#_Toc185035888)

[3.3. Тестирование системы методом черного ящика 21](#_Toc185035889)

[3.3.1. Определение тестовых сценариев 21](#_Toc185035890)

[3.3.2. Таблица тестовых случаев 21](#_Toc185035891)

[3.4. To-Do лист 23](#_Toc185035892)

[ЗАКЛЮЧЕНИЕ 25](#_Toc185035893)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc185035894)

[ПРИЛОЖЕНИЕ 1 27](#_Toc185035895)

[ПРИЛОЖЕНИЕ 2 45](#_Toc185035896)

[ПРИЛОЖЕНИЕ 3 46](#_Toc185035897)

[ПРИЛОЖЕНИЕ 4 47](#_Toc185035898)

[ПРИЛОЖЕНИЕ 5 48](#_Toc185035899)

[ПРИЛОЖЕНИЕ 6 49](#_Toc185035900)

# **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.

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**
   1. **Описание предметной области и основные процессы кадрового агентства**

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.

* 1. **Важность автоматизации процессов кадрового агентства**

Автоматизация процессов становится важным инструментом для повышения эффективности работы, минимизации ошибок и улучшения качества услуг.

Ручная обработка данных, таких как резюме кандидатов или описание вакансий, занимает значительное количество времени и часто приводит к ошибкам из-за человеческого фактора. Автоматизация позволяет ускорить рутинные процессы, такие как:

* регистрация новых кандидатов и вакансий,
* поиск подходящих кандидатов по заданным критериям,
* генерация отчетов и статистики.

Автоматизированная система мгновенно подбирает подходящих кандидатов на основе заданных фильтров.

Ручной ввод данных или их обработка могут сопровождаться ошибками, такими как:

* неверное указание контактной информации,
* несоответствие кандидатов требованиям вакансии,
* дублирование записей в базе данных.

Базы данных, созданные вручную, часто страдают от несогласованности и отсутствия структурированного подхода к хранению информации. Автоматизация обеспечивает:

* централизованное хранение данных о кандидатах, вакансиях и клиентах,
* доступ к информации через удобный интерфейс,
* возможность быстрого поиска и фильтрации данных.

Например, если менеджеру нужно найти кандидатов с определённым опытом работы, система позволяет сделать это за считанные секунды, предоставляя структурированный список с полной информацией.

Автоматизированные системы позволяют поддерживать высокий уровень взаимодействия с обеими сторонами. Например:

* кандидаты могут получать автоматические уведомления о статусе их заявки или приглашения на собеседование,
* работодатели могут получать отчёты о ходе работы по их вакансиям, включая количество найденных и отобранных кандидатов.

Это создаёт положительное впечатление о работе агентства и повышает уровень удовлетворённости клиентов.

Автоматизированные системы позволяют не только обрабатывать данные, но и анализировать их. На основе собранной информации можно:

* выявить наиболее востребованные вакансии,
* определить сезонные изменения в спросе на кадры,
* анализировать эффективность работы агентства.

Автоматизация процессов кадрового агентства повышает конкурентоспособность, улучшает качество услуг и создает устойчивый бизнес. Внедрение автоматизированных решений помогает эффективно управлять ресурсами, минимизировать ошибки и быстро реагировать на запросы клиентов и кандидатов.

* 1. **Задачи анализа данных и управления базой кадрового агентства**

Анализ данных и управление базой информации — ключевые элементы работы кадрового агентства, оптимизирующие подбор персонала и взаимодействие с клиентами. Задача заключается в сборе, структурировании и обработке данных от кандидатов и работодателей, таких как резюме, анкеты и вакансии, с приведением их к единому формату для дальнейшего анализа.

Обработка данных включает:

* + - удаление дубликатов и устаревших записей,
    - проверку полноты и корректности информации,
    - обновление данных в случае изменений (например, при смене контактной информации или появлении новых вакансий).

Одной из самых сложных задач является автоматизация подбора подходящих кандидатов под конкретные вакансии. Это требует анализа данных резюме и сравнения их с требованиями работодателей. Для этого используются алгоритмы, которые оценивают степень соответствия кандидата заданным критериям:

* уровень образования,
* профессиональные навыки,
* опыт работы,
* территориальная привязанность и готовность к переезду.

Эффективность сопоставления данных влияет на качество услуг агентства. Система должна учитывать не только точные совпадения, но и синонимы или смежные навыки. Агентство работает с постоянно обновляющимися данными, поэтому важно отслеживать изменения и сохранять историю взаимодействий.

* какие резюме уже отправлены на рассмотрение работодателю,
* какой статус присвоен каждому кандидату,
* какие вакансии остаются открытыми.

Аналитика востребованных навыков и вакансий играет ключевую роль в работе агентства. На основании собранных данных можно определить:

* какие профессии и навыки наиболее популярны в текущий момент,
* какие вакансии закрываются быстрее всего,
* существуют ли сезонные колебания спроса на определённые должности.

Эти данные полезны как для улучшения работы агентства, так и для предоставления рекомендаций клиентам. Например, аналитика может показать, что в определённый период года спрос на IT-специалистов возрастает, а на административных работников падает.

Работа с персональными данными кандидатов и клиентов накладывает на агентство обязательства по их защите. Система должна обеспечивать:

* контроль доступа к данным,
* шифрование информации при передаче и хранении,
* мониторинг и предотвращение несанкционированного доступа.

Кроме того, агентство должно соблюдать требования законодательства, такие как Федеральный закон № 152-ФЗ “О персональных данных”[[1]](#footnote-1), что требует внедрения механизмов управления доступом и удаления данных по запросу.

Для оценки эффективности работы агентства и обеспечения прозрачности перед клиентами необходимы инструменты формирования отчетов.

Отчёты позволяют не только демонстрировать клиентам результаты работы, но и выявлять внутренние проблемы и зоны для улучшения. Например, если определённый этап подбора занимает слишком много времени, это может сигнализировать о необходимости оптимизации процессов.

Современные кадровые агентства часто работают не только с внутренними базами данных, но и с внешними платформами, такими как LinkedIn[[2]](#footnote-2), порталы для поиска работы или корпоративные сайты клиентов. Задача анализа данных здесь расширяется за счёт необходимости интеграции:

* автоматический импорт и экспорт данных,
* синхронизация информации между различными системами,
* упрощение процесса обмена данными с клиентами.

Задачи анализа данных и управления базой в кадровом агентстве направлены на создание эффективной системы, способной справляться с большими объемами информации, поддерживать высокую точность и актуальность данных, а также предоставлять аналитические и отчетные инструменты. Эти процессы не только повышают внутреннюю эффективность работы агентства, но и создают добавленную ценность для клиентов, помогая им находить подходящих сотрудников в кратчайшие сроки.

1. **ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА МОДУЛЯ**
   1. **Общее назначение системы**

Система «Кадровое агентство» предназначена для автоматизации ключевых процессов управления данными о вакансиях, работодателях и кандидатах. Ее основная цель — упростить подбор подходящих кандидатов под вакансии с учетом их навыков и опыта.

* Ускорение подбора кандидатов.
* Учет данных о кандидатах и работодателях.
* Повышение точности подбора.
* Анализ эффективности работы.

Ключевые функции системы:

* Добавление кандидатов и вакансий.
* Редактирование данных.
* Просмотр данных в виде таблиц.
* Подбор кандидатов по требованиям вакансий.
* Отчеты и статистика по вакансиям, кандидатам и успешным подборам.
  1. **Описание среды разработки**

В качестве основного языка разработки был выбран **Python**, что обусловлено его универсальностью, простотой и скоростью разработки. Сравнение Python с языком С представлено в таблице:

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

Итог: Python набрал наибольшее количество баллов, что делает его идеальным выбором для создания систем автоматизации.

Для разработки системы использовались следующие библиотеки:

* PyQt5**:** для создания графического интерфейса.
* sqlite3**:** для работы с базой данных.

Эти инструменты обеспечивают реализацию всех функциональных требований системы.

* 1. **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 и работодателях для актуализации базы данных.
* Поиск кандидатов: фильтрация по ключевым параметрам для подбора подходящих кандидатов.
* Редактирование данных: внесение изменений в резюме, требования вакансий или закрытие вакансий.
* Генерация отчётов: создание отчетов о текущем состоянии базы данных, статистике и выполненных подборах.
* Сопоставление кандидатов с вакансиями: подбор подходящих кандидатов, анализируя требования вакансий и навыки соискателей.

Изображение выглядит как текст, диаграмма, снимок экрана, круг

Автоматически созданное описаниеДиаграмма прецедентов приведена на рисунке 1.

Рисунок 1. Диаграмма прецедентов (USE-CASE диаграмма)

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

* Класс Employer: представляет работодателя, хранит информацию о компании, её отрасли и описании, например, “TechCorp”, ищущую разработчиков.
* 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”.
* Взаимодействие между классами: классы связаны через общие данные. Вакансия ссылается на работодателя через employer\_id, а кандидаты сопоставляются с вакансиями по навыкам и требованиям.

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описаниеДиаграмма классов приведена на рисунке 2.

Рисунок 2. ER-диаграмма базы данных

* 1. **Выбор инструментов**

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).
* Платформонезависимость: Python-код можно запускать на различных операционных системах.

Сравнение Python с другими языками показало, что Python лучше подходит для создания гибкой системы управления данными кадрового агентства благодаря большому количеству встроенных возможностей для работы с текстовыми данными.

Сравнение языков программирования представлено в таблице 1.

|  |  |  |
| --- | --- | --- |
| Критерий | Python | C |
| Скорость разработки | Высокая благодаря простоте языка и обилию библиотек | Занимает больше времени из-за необходимости писать больше кода вручную |
| Сложность синтаксиса | Удобный и интуитивно понятный синтаксис | Требует точности и внимания к деталям |
| Базы данных | Имеются встроенные модули, например, sqlite3 | Необходимы сторонние решения и дополнительные библиотеки |
| Графический интерфейс | Легкая работа через модули, например, PyQt или Tkinter | Для создания GUI требуется использование внешних библиотек |

Таблица 1. Сравнение языков программирования

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:

* Автодополнение 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.

Дополнительные использовался SQLiteViewer**:** для визуального анализа данных в процессе разработки.

* 1. **Описание модулей**

Система «Кадровое агентство» состоит из нескольких ключевых модулей:

1. Модуль данных (модели):

* Employer: информация о работодателе (название, отрасль, описание).
* Candidate: данные о кандидате (имя, навыки, опыт).
* Vacancy: информация о вакансии (название, работодатель, требования).

1. Модуль управления базой данных (HRDatabase):

* Создание таблиц, добавление и редактирование данных (кандидатов и вакансий).
* Поиск кандидатов по навыкам и вакансий по требованиям.

1. Модуль интерфейса (HRApp):

* Управление кандидатами и вакансиями (добавление, редактирование, поиск).
* Отображение данных в таблицах.

После запуска создается объект HRApp, который управляет данными и отображает интерфейс для взаимодействия с пользователем.

Блок-схема работы представлена на рисунке 3.

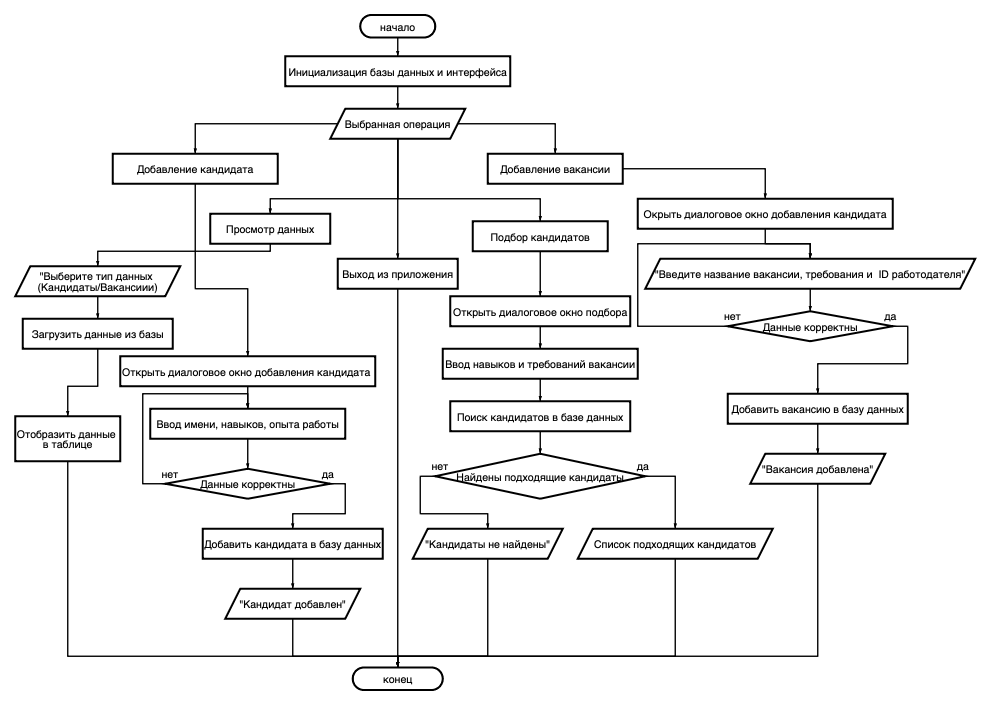


Рисунок 3. Блок-схема алгоритмов программного модуля

1. **РЕАЛИЗАЦИЯ ПРОЕКТА СИСТЕМЫ**
   1. **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.

* + 1. **Модуль моделей данных**

Этот модуль представляет основные сущности системы в виде классов:

* Employer – описывает работодателей.
* Candidate – хранит информацию о кандидатах.
* Vacancy – представляет данные о вакансиях.

Каждый класс имеет атрибуты, отражающие ключевые характеристики соответствующей сущности.

Назначение модуля: структурировать данные для работы с базой данных и интерфейсом, обеспечивать удобную работу с объектами, представляющими записи из базы данных.

* + 1. **Модуль управления базой данных**

Этот модуль, реализованный в классе HRDatabase, отвечает за все операции с базой данных SQLite. Он включает функционал для создания таблиц, выполнения CRUD-операций и сложных SQL-запросов.

Функции модуля:

1. Создание базы данных и таблиц:

При запуске системы проверяется наличие необходимых таблиц, и при их отсутствии они создаются.

1. Добавление данных:

Реализованы методы для добавления кандидатов, вакансий и работодателей.

1. Получение и фильтрация данных:

Методы добавляют новых кандидатов, вакансии и работодателей в базу данных, если их ещё нет.

1. Добавление данных:

Методы обновления позволяют вносить изменения в существующие записи.

Назначение модуля: обеспечивать надежное хранение и обработку данных, а также реализовать бизнес-логику поиска и фильтрации.

* + 1. **Модуль графического интерфейса**

Графический интерфейс реализован с использованием библиотеки PyQt5. Основной класс интерфейса — HRApp. Он отвечает за: построение окон и диалогов, взаимодействие с пользователем через кнопки, таблицы и поля ввода, а также обмен данными между пользователем и базой данных.

Основные элементы интерфейса:

1. Инициализация интерфейса:

При запуске приложения создается главное окно с приветственным сообщением и кнопками для взаимодействия с кандидатами и вакансиями.

1. Добавление и редактирование кандидатов и вакансий:

Созданы функции для добавления новых кандидатов и вакансий в базу данных через диалоговые окна. Также предусмотрена возможность редактирования существующих записей.

* 1. **Пример реализации главного модуля**

Главный модуль системы реализован в классе HRApp. Этот класс объединяет функционал всех остальных компонентов и предоставляет графический интерфейс для работы пользователя с системой. В данном

1. Конструктор класса HRApp

Конструктор выполняет инициализацию окна приложения, одключение к базе данных через объект класса HRDatabase и создание интерфейса с помощью метода init\_ui().

Конструктор подготавливает все элементы системы к работе, объединяя логику и интерфейс.

1. Создание пользовательского интерфейса

Графический интерфейс формируется с помощью метода init\_ui(). Он включает:

* Главное окно: Разделено на две области: управление кандидатами и управление вакансиями.
* Кнопки управления: Для добавления, редактирования и просмотра данных.
* Кнопка поиска: Для подбора кандидатов или вакансий.

Обеспечивает интуитивно понятный интерфейс для взаимодействия с системой.

1. Реализация функций для работы с данными

* Добавление кандидатов: функция show\_add\_candidate\_dialog открывает диалоговое окно для ввода данных кандидата, проверяет корректность ввода и передает данные в базу.
* Просмотр данных: функция show\_table отображает данные в виде таблицы.
* Поиск кандидатов: функция show\_match\_candidates позволяет находить кандидатов по Назначение модуля.

Главный модуль объединяет всю функциональность системы. Его задачи: предоставить пользователю удобный интерфейс для работы с системой, обеспечить взаимодействие между базой данных и пользователем, автоматизировать рутинные операции: добавление, поиск и редактирование данных.

* 1. **Тестирование системы методом черного ящика**
     1. **Определение тестовых сценариев**

Для тестирования методом черного ящика были разработаны тестовые сценарии, охватывающие следующие аспекты работы системы:

* Добавление данных - проверяется корректность ввода информации о кандидатах и вакансиях.
* Редактирование данных - проверяется способность системы изменять уже существующие записи.
* Поиск данных - оценивается точность поиска кандидатов по заданным навыкам.
* Отображение данных - проверяется правильность вывода информации в таблицах.
* Обработка ошибок - оценивается реакция системы на некорректный ввод.
  + 1. **Таблица тестовых случаев**

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

Система успешно прошла тестирование методом черного ящика. Все основные функции работают в соответствии с требованиями.

* 1. **To-Do лист**

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описаниеДля развития системы предлагается реализовать модуль авторизации с разграничением доступа по ролям, что улучшит безопасность. Также планируется добавление функционала для построения отчетов и визуализации данных (например, популярных навыков или успешных подборов). Для оптимизации производительности будет внедрена постраничная загрузка данных и улучшены SQL-запросы.

Рисунок 4. Прототип экранной формы интерфейса

Изображение выглядит как текст, человек, снимок экрана, одежда

Автоматически созданное описаниеИнтеграция с платформами, такими как LinkedIn и HeadHunter[[3]](#footnote-3), расширит функциональность, включая автоматическое обновление данных и взаимодействие с их API. Добавление экспорта в CSV или PDF упростит создание отчетов и обмен информацией.

Рисунок 4. Прототип сайта кадрового агентства

На рисунке 4 показан пример экранной формы интерфейса для системы «Кадрового агентства». Эта форма демонстрирует, как пользователи будут взаимодействовать с системой, например, при поиске вакансий, просмотре информации о кандидатах или добавлении новых записей в базу данных. Каждая форма представлена с соответствующими элементами интерфейса, такими как текстовые поля, кнопки для выполнения операций, выпадающие списки для выбора различных параметров и чекбоксы для многократного выбора опций. Эти элементы помогают создать интуитивно понятный и функциональный пользовательский интерфейс, облегчающий работу с данными и управлением в кадровом агентстве.

# **ЗАКЛЮЧЕНИЕ**

Разработка курсового проекта «Кадровое агентство» позволила создать систему, автоматизирующую ключевые процессы управления данными о вакансиях, кандидатах и работодателях. Проект объединяет современные технологии программирования, тестирования и оптимизации, обеспечивая функциональное и надежное приложение.

Основной модуль системы, реализованный в классе HRApp, обеспечивает интуитивно понятный графический интерфейс, который позволяет пользователям легко взаимодействовать с данными. Интерфейс включает функции добавления, редактирования, поиска и отображения информации, что делает приложение универсальным инструментом для HR-специалистов. Связь интерфейса с базой данных, реализованная через класс HRDatabase, обеспечивает безопасное и эффективное хранение данных. Реализация SQL-запросов и продуманный подход к структуре таблиц базы данных позволяют системе обрабатывать большие объемы информации с высокой производительностью.

Тестирование системы с использованием метода черного ящика показало стабильность работы и соответствие требованиям. Были разработаны тестовые сценарии, охватывающие все аспекты функционирования системы.

Проект заложил основу для дальнейшего развития, включая возможную реализацию модуля авторизации для повышения безопасности. Курсовой проект продемонстрировал значимость комплексного подхода к проектированию, разработке и тестированию программного обеспечения.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Стандарты и законодательные материалы

1. ГОСТ 7.1–2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления».
2. ГОСТ 7.32–2001 «Отчет о научно-исследовательской работе. Структура и правила оформления»
3. ГОСТ 7.80–2000 «Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления»
4. ГОСТ 19.701-90 (ИСО 5807-85) — «Единая система программной документации. Схемы алгоритмов, программ, данных и систем».
5. Федеральный закон от 27 июля 2006 г. N 152-ФЗ  
   "О персональных данных"

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.

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>

**ПРИЛОЖЕНИЕ 1.** Код kadr.py

import sqlite3  
from PyQt5.QtWidgets import (QApplication, QMainWindow, QVBoxLayout, QHBoxLayout, QWidget, QLabel, QLineEdit,  
 QPushButton, QTableWidget, QTableWidgetItem, QMessageBox, QDialog, QInputDialog)  
from PyQt5.QtCore import Qt  
  
# Модели данных для кандидатов, работодателей и вакансий  
class Employer:  
 def \_\_init\_\_(self, name, industry, description):  
 self.name = name # Название работодателя  
 self.industry = industry # Отрасль  
 self.description = description # Описание работодателя  
  
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  
  
# Класс для работы с базой данных  
class HRDatabase:  
 def \_\_init\_\_(self, db\_name="кадровое агенство 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 Employers (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 industry TEXT NOT NULL,  
 description TEXT)''')  
 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 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()  
  
# Главный класс приложения  
class HRApp(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Кадровое агентство") # Устанавливаем заголовок окна  
 self.db = HRDatabase() # Создание объекта базы данных  
 self.init\_ui() # Инициализация пользовательского интерфейса  
  
 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\_()  
  
 #данные для заполнения базы  
 candidates = [  
 Candidate("Сельгира Сангаджи-Горяева", "Python, Django, SQL", 5),  
 Candidate("Мария Смирнова", "JavaScript, React, HTML, CSS", 3),  
 Candidate("Алексей Кузнецов", "Java, Spring, MySQL", 4),  
 Candidate("Елена Попова", "C++, Qt, OpenGL", 7),  
 Candidate("Дмитрий Сидоров", "Python, Flask, Docker", 6),  
 Candidate("Анна Петрова", "Project Management, Agile, Scrum", 8),  
 Candidate("Максим Волков", "JavaScript, Node.js, MongoDB", 2),  
 Candidate("Светлана Николаева", "Python, Machine Learning, TensorFlow", 4),  
 Candidate("Ольга Васильева", "PHP, Laravel, SQL", 5),  
 Candidate("Григорий Крылов", "C#, .NET, Azure", 3),  
 Candidate("Наталья Орлова", "Data Analysis, Python, SQL", 6),  
 Candidate("Павел Михайлов", "DevOps, AWS, Kubernetes", 5),  
 Candidate("Виктория Сергеева", "UI/UX Design, Figma, Adobe XD", 2),  
 Candidate("Николай Романов", "Java, Android Development", 4),  
 Candidate("Ирина Шестакова", "Marketing, SEO, Google Analytics", 7)  
 ]  
  
 vacancies = [  
 Vacancy("Backend-разработчик", 1, "Python, Django, SQL"),  
 Vacancy("Frontend-разработчик", 2, "JavaScript, React, HTML, CSS"),  
 Vacancy("Java-разработчик", 3, "Java, Spring, MySQL"),  
 Vacancy("C++ разработчик", 4, "C++, Qt, OpenGL"),  
 Vacancy("DevOps-инженер", 5, "DevOps, AWS, Kubernetes"),  
 Vacancy("Data Scientist", 6, "Python, Machine Learning, TensorFlow"),  
 Vacancy("PHP-разработчик", 7, "PHP, Laravel, SQL"),  
 Vacancy("Project Manager", 8, "Project Management, Agile, Scrum"),  
 Vacancy("UI/UX Designer", 9, "UI/UX Design, Figma, Adobe XD"),  
 Vacancy("Mobile-разработчик", 10, "Java, Android Development"),  
 Vacancy("Маркетолог", 11, "Marketing, SEO, Google Analytics"),  
 Vacancy("Системный аналитик", 12, "Data Analysis, Python, SQL"),  
 Vacancy("Cloud Engineer", 13, "C#, .NET, Azure"),  
 Vacancy("Fullstack-разработчик", 14, "JavaScript, Node.js, MongoDB"),  
 Vacancy("Python-разработчик", 15, "Python, Flask, Docker")  
 ]  
  
 # 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 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(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([])  
 window = HRApp()  
 window.show()  
 app.exec\_()

# Изображение выглядит как текст, диаграмма, снимок экрана, линия Автоматически созданное описание**ПРИЛОЖЕНИЕ 2.** Блок-схема алгоритма главного модуля

# **ПРИЛОЖЕНИЕ 3.** Блок-схема модуля «добавление кандидата»

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

# **ПРИЛОЖЕНИЕ 4.** Блок-схема модуля «просмотр данных»

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

# **ПРИЛОЖЕНИЕ 5.** Блок-схема модуля «добавление вакансии»

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

# **ПРИЛОЖЕНИЕ 6.** Блок-схема модуля «поиск кандидатов/вакансий»

Изображение выглядит как диаграмма, линия, текст, План

Автоматически созданное описание

1. Федеральный закон "О персональных данных" от 27.07.2006 N 152-ФЗ [↑](#footnote-ref-1)
2. Социальная сеть для поиска и установления деловых контактов [↑](#footnote-ref-2)
3. крупнейшая российская компания интернет-рекрутмента, развивающая бизнес в России, Белоруссии, Казахстане. [↑](#footnote-ref-3)