

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего
образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 1 _

д.ф.-м.н., доцент
должность, уч. степень, звание

С.А. 11.06.25
подпись, дата

А.О. Смирнов
инициалы, фамилия

БАКАЛАВРСКАЯ РАБОТА

на тему Моделирование процессов подбора и оценки персонала в кадровом
менеджменте

выполнена Поспеловой Анастасией Александровной
фамилия, имя, отчество студента в творительном падеже

по направлению подготовки 01.03.02 Прикладная математика и информатика
код наименование направления

направленности 01 Прикладная математика и информатика в
код наименование направленности
наукоемком производстве
наименование направленности

Студент группы № M112 А.А. Поспелова
подпись, дата инициалы, фамилия

Руководитель
профессор, д.т.н., доцент *Л.П.* 05.06.2025 Л.П. Вершинина
должность, уч. степень, звание подпись, дата инициалы, фамилия

*В работе не содержится информации
с ограниченным доступом, и отсутствуют
сведения, представляющие коммерческую ценность.*

Санкт-Петербург 2025

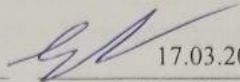
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой № 1

д.ф.-м.н., доцент

должность, уч. степень, звание



17.03.2025

подпись, дата

А.О. Смирнов

инициалы, фамилия

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

студенту группы №

M112

Поспеловой Анастасии Александровне

(фамилия, имя, отчество)

на тему

Моделирование процессов подбора и оценки персонала

в кадровом менеджменте

утвержденную приказом ГУАП от 14.03.2025

№ 11-308/25

Цель работы: повышение эффективности подбора персонала путем

оптимизации и автоматизации процесса оценивания кандидатов на должность

Задачи, подлежащие решению: провести анализ существующих подходов к подбору и оценке

персонала в кадровом менеджменте и выявить проблемы; предложить методы,

позволяющие повысить эффективность процесса подбора кандидата на должность; разработать

информационную систему оценки кандидатов на должность в IT-индустрии.

Содержание работы (основные разделы): 1. Процесс подбора и оценки персонала в

кадровом менеджменте. 2. Анализ методов оценки кандидатов в IT-индустрии.

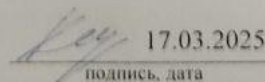
3. Разработка информационной системы оценки кандидатов на должность в IT-индустрии.

Срок сдачи работы « 05 » июня 2025

Руководитель

профессор, д.т.н., доцент

должность, уч. степень, звание



17.03.2025

подпись, дата

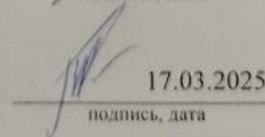
Л.П. Вершинина

инициалы, фамилия

Задание принял к исполнению

студент группы №

M112



17.03.2025

подпись, дата

А.А. Поспелова

инициалы, фамилия

СОДЕРЖАНИЕ

| | |
|--|----|
| Введение | 4 |
| 1 Процесс подбора и оценки персонала в менеджменте | 7 |
| 1.1 Проблемы процесса подбора персонала | 7 |
| 1.2 Существующие методы подбора персонала | 11 |
| 1.3 Тестирование кандидатов в IT-индустрии | 16 |
| 2 Анализ методов оценки персонала в IT-индустрии | 20 |
| 2.1 Метод нечётких деревьев | 20 |
| 2.2 Метод анализа иерархий | 22 |
| 2.3 Метод бинарного выбора | 25 |
| 2.4 Сравнительный анализ методов | 27 |
| 3 Разработка информационной системы подбора и оценки кандидатов на должность в IT-индустрии | 30 |
| 3.1 Разработка базы данных | 30 |
| 3.2 Применяемые тесты для проведения оценки кандидатов | 33 |
| 3.3 Программная реализация приложения | 39 |
| ЗАКЛЮЧЕНИЕ | 61 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 63 |
| ПРИЛОЖЕНИЕ 1 | 65 |
| ПРИЛОЖЕНИЕ 2 | 70 |
| ПРИЛОЖЕНИЕ 3 | 88 |

Введение

В современном быстро развивающемся мире ежегодно происходит множество изменений. Технологический прогресс неумолимо движется вперёд, всё сильнее влияя на мир бизнеса. Компании, в погоне за наибольшей прибылью, следят за инновациями и адаптируются к ним. Для поддержания собственной конкурентоспособности организации стараются собрать и удержать востребованный рабочий коллектив. В связи с чем ежегодно наблюдается тенденция к усложнению процессов найма в IT-индустрии, особенно на фоне неуклонно растущего количества кандидатов. В таких условиях процессы подбора и оценки персонала требуют внедрения технологий, позволяющих оптимизировать затрачиваемое время и ресурсы на поиск максимально подходящих должности людей.

Степень разработанности проблемы.

Говоря об истории развития теории управления персоналом с применением математических методов, в первую очередь, стоит упомянуть школу количественных методов. Данное направление сформировалось в 50-е годы XX века. В то время школа возникла благодаря развитию математических наук, статистики, информатики и компьютерной техники. Наиболее наглядными примерами использования достижений количественной школы стали модели оценки социально-экономической эффективности управления персоналом и модели, базирующиеся на теории вероятностей, теории игр и нейролингвистическом программировании. Представителями школы можно назвать Леонида Витальевича Канторовича («Экономический расчёт наилучшего использования ресурсов», 1959), Виктора Валентиновича Новожилова («Проблемы измерения затрат и результатов при оптимальном планировании», 1967), Людвиг фон Берталанфи («Системное видение человека», 1981), Рассела Акоффа («Введение в исследование операций», 1957), Артура Гольдбергера («Экономические и механические модели передачи между поколениями», 1989).

На сегодняшний день разработка и применение точных методов для анализа кадров становится все более востребованным в сфере управления. Сегодня одним из лидеров в этой области можно назвать Дмитрия Новикова. Его работы посвящены разработке математических моделей управления социально-экономическими системами. Например, «Теория управления организационными системами» (2005) — фундаментальный труд, охватывающий широкий спектр вопросов управления организационными системами с применением математических методов. Ещё одним ярким примером можно назвать Владимира Буркова — профессора, доктора технических наук, специалиста в области теории активных систем и механизмов управления экономическими системами; автора работы "Математические основы управления организацией" (2003).

Целью дипломного исследования является повышение эффективности подбора персонала путём оптимизации и автоматизации процесса оценивания кандидатов на должность.

Для достижения указанной цели в работе поставлены и решены следующие *задачи*:

- провести анализ существующих подходов к подбору и оценке персонала в кадровом менеджменте и выявить проблемы;
- предложить методы, позволяющие повысить эффективность процесса подбора кандидата на должность;
- разработать информационную систему оценки кандидатов на должность в IT-индустрии.

Объектом дипломного исследования является процесс подбора и оценки персонала в компании.

Предмет исследования составляют математические и аналитические методы для оптимизации принятия решения при подборе персонала.

Методология и методы исследования.

В работе использован метод анализа иерархий.

Теоретическая значимость результатов.

Теоретическая значимость работы заключается во вкладе в развитие теоретических основ управления человеческими ресурсами, демонстрируя возможности применения информационной системы для оптимизации процессов подбора и оценки персонала.

Практическая значимость работы.

Практическая значимость результатов состоит в том, что разработанная информационная система автоматизирует процесс тестирования и позволяет получить объективное представление о компетенциях кандидатов.

Структура и содержание: работа состоит из введения, трёх глав, заключения, списка использованных источников и приложения. В первой главе рассмотрены проблемы и существующие модели процесса подбора и оценки персонала. Акцентируется внимание на тестировании кандидатов в IT-сфере. Во второй главе проведён анализ и сравнение методов, используемых для оценки персонала. В третьей главе представлен процесс создания информационной системы для тестирования кандидатов на должность в IT-индустрии. В заключении сделаны выводы, относящиеся к работе в целом.

1 Процесс подбора и оценки персонала в менеджменте

1.1 Проблемы процесса подбора персонала

Процесс подбора персонала является критически важной функцией для любой организации, напрямую влияющей на её конкурентоспособность и долгосрочный успех. Говоря о персонале, будем иметь в виду социально-экономическую категорию, обозначающую личный состав организации, работающий по найму и обладающий определёнными признаками: квалификацией, компетенцией, способностями, установками [1]. Кадровый менеджмент – это целенаправленная деятельность системы управления персоналом, направленная на развитие и эффективное использование трудовых ресурсов [2]. Эффективный подбор квалифицированных сотрудников обеспечивает компании необходимые компетенции для достижения стратегических целей. Однако данный процесс часто сталкивается с рядом проблем, которые могут негативно сказаться на качестве нанимаемых сотрудников, временных затратах и финансовых издержках.

В настоящее время на рынке наблюдается острая нехватка квалифицированных кандидатов на вакансии. На 2024 год уровень безработицы в России составлял 1 914,9 тыс. чел. или 2,52% от трудоспособного населения [3]. Дефицит кадров коснулся всех отраслей в большинстве регионов страны. Первостепенной причиной можно назвать демографическую яму 90-х – нулевых годов, когда уровень рождаемости упал ниже уровня смертности, и сейчас кадровый менеджмент столкнулся с последствиями. Ещё одной причиной послужило то, что кандидаты охотнее устраиваются на менее квалифицированную работу, если на ней нужно меньше работать и степень ответственности остаётся минимальной. Это в определённой степени выглядит более привлекательно для соискателей, из-за чего степень дефицита кадров усугубляется. Также невозможность повышения квалификации может быть связана с недостаточностью образования сотрудников, что может быть серьёзным препятствием на пути к увеличению числа квалифицированных

рабочих, что в последствии сказывается на развитии компании. Эта проблема решается с помощью создания систем внутреннего обучения и переподготовки [4]. Однако стоит отметить, что из-за крайне низкой конкуренции на рынке труда кандидаты теряют желание развиваться и повышать квалификацию. И внаслідок, нельзя не упомянуть эмиграцию молодых специалистов в 2022-2023 годах. Среди них высокая доля IT-специалистов, которых сейчас так не хватает.

Нехватка квалифицированных кандидатов серьёзно отражается на процессе подбора персонала, поскольку количества людей, обладающих необходимыми навыками, может быть недостаточно. Компании вынуждены нести большие денежные и временные затраты в процессе рекрутинга на поиск подходящего человека. Также в условиях низкой конкуренции в определённой сфере зарплатные ожидания специалистов значительно возрастают.

Правильная организация системы подбора и оценки кандидатов играет важную роль в процессе поиска персонала. Здесь можно сказать о том, что заказчики зачастую сами до конца не понимают, кто им нужен. А при составлении профиля должности возникает несогласованность с менеджерами по персоналу. Тем временем руководители ожидают немедленных результатов поиска специалиста, хотя на деле требуется время [5].

Чтобы регулировать процесс поиска идеального кандидата, разрабатываются строгие стандарты подбора, которым неукоснительно следуют и руководители, и менеджеры. Для этого процесса внедряется единый, отлаженный механизм принятия решений. Однако при оценке кандидатов, а также при вынесении вердикта о пригодности соискателя, неизбежно проявляется субъективность оценивающего, внося свой отпечаток в окончательное решение. Ещё одной из наиболее распространённых проблем является нечёткое или неполное определение требований к кандидату. Отсутствие детализированного профиля должности, включающего необходимые навыки, знания, опыт и личностные качества, приводит к

привлечению нерелевантных резюме и увеличению времени на первичный отбор. Чёткое определение требований к должности является отправной точкой для успешного процесса рекрутинга [6]. Неточности в описании вакансии также могут отпугнуть потенциально квалифицированных кандидатов, которые не уверены, соответствуют ли они требованиям.

Рассмотрим вопрос затянутости процесса подбора персонала. Данная проблема приводит к потере талантливых кандидатов, снижению продуктивности и, в конечном счёте, негативному влиянию на имидж компании как работодателя. Многочисленные факторы могут способствовать затягиванию процесса подбора персонала. Среди наиболее распространённых выделяется уже обозначенная проблема нечёткого определения требований к вакансии. Размытое описание должностных обязанностей, неточные критерии оценки кандидатов, отсутствие согласованности между нанимающим менеджером и руководителем – всё это приводит к увеличению времени на поиск и оценку подходящих кандидатов. Также для размещения вакансий иногда используют устаревшие или непопулярные платформы, при этом не уделяя внимания активному рекрутингу в социальных сетях. В таких условиях наниматель сильно ограничивает распространение информации о вакансии, тем самым замедляя процесс подбора.

Рассматривая проблему затянутости процесса подбора персонала, важно отметить, что работодатели порой перегибают палку на этапе оценки кандидатов избыточным количеством собеседований и использованием устаревших методик. Большое количество проверок кандидата требует значительных временных затрат на проведение собеседований, обработку результатов и оценку. Как правило, кандидаты не ограничиваются вакансией в одной компании, следовательно, за то время, которое уходит на все этапы найма, кандидат может найти другое место работы или демотивироваться длительным ожиданием и отказаться от дальнейших этапов проверки.

Ещё одной ключевой проблемой, приводящей к затянутости подбора персонала, можно назвать отсутствие автоматизированных систем управления рекрутингом. Данная проблема представляет собой значительный вызов для современных организаций, стремящихся к оптимизации процессов найма квалифицированных специалистов. В условиях динамично меняющегося рынка труда и возрастающей конкуренции за таланты, ручные методы рекрутинга становятся неэффективными, приводя к увеличению временных затрат и снижению качества принимаемых решений.

Одним из ключевых аспектов проблемы является трудоёмкость процесса. Рекрутеры вынуждены тратить много времени на рутинные административные задачи, такие как просмотр резюме, согласование графиков собеседований, ведение документации и т.д. Автоматизация этих задач позволяет значительно сократить время, затрачиваемое на каждый этап рекрутинга, и перенаправить ресурсы на более стратегические направления.

Вторым важным аспектом является снижение качества подбора кандидатов. Без использования автоматизированных инструментов (например, системы отслеживания кандидатов (ATS), платформы для оценки компетенций), рекрутеры вынуждены полагаться на субъективные оценки и неполную информацию о кандидатах. Это может приводить к ошибкам при принятии итогового решения о найме, что впоследствии способствует увеличению текучести кадров. С помощью подобных инструментов можно систематизировать процесс отбора, а также автоматизировать проверку резюме на соответствие требованиям вакансии. Автоматизированные системы позволяют собирать и анализировать данные о каждом этапе рекрутинга, предоставляя рекрутерам и руководителям информацию, необходимую для принятия обоснованных решений, поскольку такие системы проводят объективную оценку компетенций кандидатов. И в целом упрощается процесс анализа. Можно легко отслеживать время закрытия вакансий, стоимость найма, источники привлечения наиболее квалифицированных кандидатов и причины

ухода сотрудников в течение первого года работы. В конечном счёте, автоматизация упомянутых этапов рекрутинга, а также сторонних процессов, связанных с менеджментом, приводит к значительному сокращению временных затрат отдела кадров.

1.2 Существующие методы подбора персонала

Оптимизация подбора требует всестороннего анализа существующих методов, выявления их преимуществ и недостатков, а также адаптации к специфике конкретной организации и рыночным условиям. Традиционно методы подбора персонала классифицируются на основе различных критериев, таких как источники привлечения кандидатов, инструменты оценки и этапы принятия решений.

Одним из наиболее распространённых подходов является классификация методов подбора по источникам привлечения кандидатов. В соответствии с этим подходом выделяют внутренние и внешние методы. Внутренние методы, включающие продвижение по службе, ротацию кадров и привлечение сотрудников по рекомендациям, позволяют использовать потенциал существующего персонала, повышать мотивацию и лояльность сотрудников, а также снижать затраты на поиск и адаптацию новых работников. Однако внутренние методы могут ограничивать приток новых идей и компетенций, а также приводить к конфликтам интересов и снижению морального духа в коллективе.

Внешние методы, в свою очередь, предполагают привлечение кандидатов из внешних источников, таких как специализированные сайты по поиску работы, кадровые агентства, социальные сети и образовательные учреждения. Внешние методы позволяют расширить круг потенциальных кандидатов, привлечь специалистов с уникальными компетенциями и привнести новые знания и опыт в организацию. Большинство организаций для этой цели используют онлайн-ресурсы для поиска кандидатов. Тем не менее внешние

методы сопряжены с более высокими затратами на поиск и отбор, а также требуют больше времени на адаптацию новых сотрудников.

Другим способом дифференциации критериев может быть классификация по этапам принятия решений. В соответствии с этим критерием выделяют методы, используемые на этапе привлечения кандидатов, методы, используемые на этапе отбора, и методы, используемые на этапе адаптации. На этапе привлечения кандидатов используются методы, направленные на формирование положительного имиджа работодателя и привлечение внимания потенциальных кандидатов. К ним можно отнести участие в ярмарках вакансий, размещение рекламы в СМИ и социальных сетях, а также развитие бренда работодателя.

На этапе подбора используются методы, направленные на оценку соответствия кандидатов требованиям должности и выбор наиболее подходящего кандидата. Инструменты оценивания персонала подразделяются на две категории: общепринятые и специализированные (или стандартные и нестандартные). Общепринятые методы широко распространены среди работодателей для оценки большинства позиций и хорошо знакомы кандидатам. Специализированные методы применяются ситуативно, преимущественно для подбора персонала определённой квалификации, и могут вызывать различную реакцию со стороны кандидатов. Граница между стандартными и нестандартными методами оценки может отражать лишь субъективное понимание работодателей и соискателей и зависит скорее от частоты использования тех или иных методов в целом [7].

На этапе адаптации используются методы, направленные на интеграцию нового сотрудника в коллектив и его обучение необходимым знаниям и навыкам. Эффективно выстроенная система адаптации персонала в компании способствует минимизации временных затрат как рабочей группы, так и руководителя на интеграцию нового работника. Это позволяет разработать проверенный алгоритм введения в должность вновь прибывших специалистов,

не оказывая негативного воздействия на общую производительность отдела, подразделения и организации в целом. Методы адаптации можно разделить на три категории: методы обучения, вовлечения и сопровождения. На этапе обучения адаптация сотрудника реализуется в форме различных тренингов. К группе методов вовлечения относят, например, встречи с новыми сотрудниками компании, экскурсии, мини-совещания, деловые игры. Для реализации этапа сопровождения подойдёт использование таких методов, как наставничество, координация со стороны руководителя, дистанционное сопровождение [8].

Другим важным критерием классификации методов оценки является инструментарий оценки кандидатов. Эффективный инструментарий оценки кандидатов является критически важным компонентом успешной стратегии управления человеческими ресурсами. Он позволяет организациям принимать обоснованные решения о найме, основанные на объективных данных и прогнозах успешности кандидата в конкретной роли. Использование валидированных и надёжных инструментов оценки повышает вероятность найма высокоэффективных сотрудников, снижает текучесть кадров и оптимизирует расходы на подбор персонала.

В основе любого инструментария оценки кандидатов лежит чёткое определение требуемых компетенций для конкретной должности. Компетенции представляют собой совокупность знаний, навыков, умений и личностных характеристик, необходимых для успешного выполнения рабочих задач. Оценка компетенций предполагает использование различных методов, направленных на выявление и измерение этих характеристик у кандидатов.

В соответствии с этим критерием выделяют методы, основанные на анализе резюме и сопроводительных писем, интервью, тестировании, оценке профессиональных навыков и проверке рекомендаций. Анализ резюме и сопроводительных писем позволяет получить первоначальное представление об образовании, опыте работы и квалификации кандидатов. Однако в полной мере

оценить личностные качества и потенциал кандидатов только на основе этой информации не представляется возможным.

Интервью, как структурированные, так и неструктурированные, являются одним из наиболее распространённых методов оценки кандидатов. Структурированные интервью, основанные на заранее разработанном наборе вопросов, позволяют обеспечить сопоставимость оценок кандидатов и снизить влияние субъективных факторов. Неструктурированные интервью, напротив, предоставляют больше свободы для импровизации и позволяют получить более глубокое понимание личностных качеств и мотивации кандидатов. Однако неструктурированные интервью более подвержены субъективным оценкам и могут быть менее надёжными.

Тестирование, включающее психологические тесты, тесты на профессиональные знания и навыки, или, например, тесты на когнитивные способности, позволяет оценить различные аспекты потенциала кандидатов и спрогнозировать их успешность в работе. Психологические тесты, такие как тесты на личностные качества и мотивацию, позволяют оценить соответствие кандидата корпоративной культуре и требованиям должности. Тесты на профессиональные знания и навыки позволяют оценить уровень владения необходимыми компетенциями, а тесты на когнитивные способности – способность к обучению и решению проблем.

Оценка профессиональных навыков, включающая выполнение тестовых заданий и презентацию проектов, позволяет оценить практические навыки и умения кандидатов в условиях, максимально приближенных к реальной работе. Данный метод особенно эффективен для оценки кандидатов на должности, требующие высокой квалификации и опыта. Проверка рекомендаций, в свою очередь, позволяет получить дополнительную информацию о кандидате от его предыдущих работодателей и коллег. Правда, проверка рекомендаций может быть затруднена из-за конфиденциальности информации и нежелания предыдущих работодателей давать негативные отзывы.

Задачу подбора персонала можно также рассматривать как многокритериальную задачу принятия решений [9]. Для оптимизации выбора в условиях множества конкурирующих критериев, задачи многокритериального принятия решений предоставляют структурированный подход. Решение базируется на создании экономико-математической модели, которая обеспечивает аналитическую основу для формализации процесса выбора и определения наилучшего варианта. Существует большое количество методов решения задач по выбору лучшей альтернативы: метод суммы мест, метод взвешенной суммы, метод сумм взвешенных значений критериев, метод TOPSIS, метод ELECTRE I и др. Каждый метод уникален и имеет свои особенности. Достоверность полученного решения проводится с помощью оценки чувствительности приоритетности альтернатив к изменению весов критериев [10].

Решение задачи подбора персонала зачастую требует системного подхода [11]. Сам процесс включает в себя множество элементов сложными связями, что делает процедуру принятия решения многоступенчатой. Для рационального выбора в таких условиях можно рассмотреть применение экспертных систем (ЭС). ЭС моделируют интеллектуальную деятельность специалистов, основываясь на алгоритмах, которые позволяют обрабатывать данные и обучаться на их основе. Знания экспертов становятся основой правил и алгоритмов таких систем, за счёт чего решаются сложные задачи по принятию решений и разработке рекомендаций, в том числе в вопросе подбора персонала. Для создания ЭС детально изучается предметная область и собираются экспертные данные. Затем данные подвергаются формализации: сведения структурируются и представляются с помощью математических моделей, правил и логических связей. В конечном счёте получается система, способная с повышенной точностью предоставить обоснованные решения, при этом минимизируя вероятность ошибок.

Анализ существующих методов подбора персонала позволяет организациям выбирать наиболее эффективные инструменты и подходы, соответствующие их специфическим потребностям и задачам. Комплексный подход к подбору персонала, учитывающий как внутренние, так и внешние источники, а также использующий широкий спектр инструментов оценки, позволяет организациям привлекать и отбирать квалифицированных и мотивированных сотрудников, способных внести значительный вклад в достижение стратегических целей.

1.3 Тестирование кандидатов в IT-индустрии

В IT-индустрии, где конкуренция за квалифицированные кадры чрезвычайно высока, тестирование кандидатов на должность выступает в качестве критически важного метода оценки, позволяющего компаниям выявлять наиболее перспективных специалистов, обладающих необходимыми техническими навыками, когнитивными способностями и личностными качествами, необходимыми для достижения успеха в быстро меняющейся технологической среде. Этот метод, при условии его корректного применения, значительно повышает эффективность процесса найма, снижает риски, связанные с неправильным подбором персонала, и способствует формированию высокопроизводительных команд.

Тестирование кандидатов как метод оценки включает в себя использование различных инструментов и техник, направленных на измерение определённых компетенций, необходимых для выполнения конкретной роли в IT-компании. Эти инструменты варьируются от стандартных тестов на знание языков программирования и алгоритмов до сложных симуляций, имитирующих реальные рабочие ситуации. Целью тестирования является получение объективных данных о способностях кандидата, которые дополняют информацию, полученную в ходе собеседований и анализа резюме.

Одним из наиболее распространённых видов тестирования в IT является техническое тестирование, которое включает в себя проверку знаний и навыков кандидата в конкретных областях, таких как разработка программного обеспечения, веб-разработка, анализ данных или кибербезопасность. Технические тесты могут проводиться в различных форматах, включая онлайн-тесты, задачи на написание кода, практические задания и тесты на знание конкретных технологий и инструментов. Выбор конкретного формата зависит от специфики должности и требуемых навыков.

Помимо технических навыков, важное значение имеет оценка когнитивных способностей кандидатов, таких как логическое мышление, аналитическое мышление и способность к решению проблем. Когнитивные тесты, такие как тесты на абстрактное мышление, числовые тесты и вербальные тесты, позволяют оценить общую интеллектуальную подготовку кандидата и его способность к обучению и адаптации к новым задачам. В IT-индустрии, где постоянно возникают новые технологии и задачи, когнитивные способности играют ключевую роль в успешной адаптации и развитии специалиста.

Оценка личностных качеств кандидатов также является важным аспектом тестирования, особенно при формировании команд и назначении на руководящие должности. Личностные тесты, такие как опросники личности, позволяют оценить такие характеристики, как коммуникабельность, ответственность, стрессоустойчивость, умение работать в команде и лидерские качества. Важно отметить, что личностные тесты не должны использоваться для дискриминации кандидатов, а результаты должны интерпретироваться с осторожностью и в контексте других данных о кандидате.

Эффективность тестирования кандидатов в IT-индустрии зависит от нескольких факторов, включая валидность и надёжность используемых инструментов, соответствие тестов требованиям вакансии, а также правильную интерпретацию результатов. Валидность тестов показывает, насколько хорошо они измеряют то, что должны измерять, а надёжность показывает, насколько

стабильны и воспроизводимы результаты. Использование валидированных и надёжных тестов повышает вероятность принятия обоснованных решений о найме и снижает риск найма некомпетентных сотрудников.

Разработка и внедрение эффективной системы тестирования кандидатов требует значительных инвестиций в разработку и валидацию тестов, а также в обучение персонала, проводящего оценку. Однако эти инвестиции оправдываются за счёт повышения качества найма, снижения текучести кадров и повышения производительности труда. В условиях острой конкуренции за квалифицированные кадры в IT-индустрии, тестирование кандидатов на должность является необходимым инструментом для компаний, стремящихся к успеху и инновациям.

Несмотря на очевидные преимущества, использование тестирования кандидатов сопряжено с рядом этических и юридических вопросов. Важно обеспечить конфиденциальность данных кандидатов, избегать дискриминации и соблюдать требования законодательства в области защиты персональных данных. Также необходимо предоставлять кандидатам обратную связь по результатам тестирования и давать возможность обжаловать результаты.

Тем не менее, в данной работе мы подробно рассматриваем тестирование как основной метод подбора и оценки кандидата для получения представления о его компетенциях. Как было упомянуто в пункте 1.1, автоматизация системы оценивания является одной из ключевых задач кадрового отдела, поскольку за счёт этого оптимизируется множество рабочих аспектов. То же можно отметить и в отношении тестирования. Одним из ключевых аспектов автоматизации является возможность проведения тестирования с использованием ПК, что позволяет охватить широкий географический диапазон кандидатов и существенно сократить время, необходимое для проведения оценки. Онлайн-тестирование предоставляет кандидатам возможность проходить тесты в удобное для них время и месте, что повышает их вовлеченность и снижает стресс, связанный с процессом оценки. Кроме того, автоматизированные

системы позволяют мгновенно обрабатывать результаты тестов и предоставлять подробные отчёты о компетенциях и навыках кандидатов, что значительно упрощает процесс принятия решений о найме.

Автоматизация процесса тестирования также позволяет повысить объективность оценки, минимизируя влияние субъективных факторов, таких как личные предпочтения интервьюера или предвзятость, обусловленная социокультурными стереотипами. Автоматизированные системы оценки результатов тестов используют чёткие и стандартизированные критерии, что позволяет сравнивать кандидатов по единым показателям и выявлять наиболее перспективных специалистов. Это особенно важно при оценке технических навыков, где объективные данные о знаниях и умениях кандидата имеют решающее значение.

2 Анализ методов оценки персонала в IT-индустрии

2.1 Метод нечётких деревьев

Традиционные методы оценки, основанные на субъективных мнениях экспертов и формализованных тестовых заданиях, часто оказываются недостаточными для всесторонней характеристики потенциального сотрудника, особенно учитывая многогранность компетенций, требуемых в современной IT-сфере. В связи с этим возрастает актуальность применения методов искусственного интеллекта, способных учитывать неопределённость и нечёткость оценок, а также моделировать логические связи между различными критериями оценки.

Одним из перспективных подходов к решению данной задачи является использование метода нечётких деревьев (Fuzzy Decision Trees, FDT) [12]. FDT представляют собой расширение классических деревьев решений, в которых узлы, ветви и листья могут содержать нечёткие множества, описывающие степень принадлежности к определённому классу или категории. Это позволяет учитывать неопределённость и субъективность экспертных оценок, а также моделировать сложные взаимосвязи между различными критериями оценки кандидатов.

Преимущества использования FDT для оценки кандидатов в IT:

- **Обработка неопределённости:** в отличие от классических методов, FDT способны работать с нечёткими и размытыми данными, что позволяет учитывать субъективные мнения экспертов и неполную информацию о кандидате.
- **Моделирование сложных взаимосвязей:** FDT позволяют представлять сложные логические правила, связывающие различные критерии оценки, такие как навыки программирования, опыт работы, коммуникативные навыки и личностные качества.

- Интерпретируемость: структура дерева решений позволяет визуализировать процесс оценки и понять, какие критерии оказывают наибольшее влияние на итоговое решение.
- Адаптивность: FDT могут быть обучены на основе исторических данных о кандидатах и результатах их работы, что позволяет адаптировать модель к специфическим требованиям компании и конкретной должности.
- Автоматизация: процесс оценки кандидатов может быть автоматизирован, что снижает трудозатраты и повышает объективность отбора.

Процесс построения FDT для оценки кандидатов на должность в IT будет выглядеть следующим образом:

На первом этапе необходимо определить ключевые критерии оценки, которые важны для успешного выполнения работы на конкретной должности. Например, для разработчика программного обеспечения это могут быть: знание языков программирования (например, Python, Java, C++), опыт работы с различными фреймворками и библиотеками, навыки алгоритмизации и структур данных, опыт командной работы, коммуникативные навыки и знание английского языка. Далее для каждого критерия необходимо определить лингвистические переменные, описывающие степень его выраженности. Например, для критерия "знание языка программирования Python" можно использовать следующие лингвистические переменные: "низкое", "среднее", "высокое", "отличное". Каждой лингвистической переменной соответствует нечёткое множество, определяющее степень принадлежности кандидата к данной категории. После этого необходимо собрать данные о кандидатах, которые будут использоваться в качестве значений критериев оценки. На основе собранных данных строится дерево решений, в котором каждый узел соответствует одному из критериев оценки, а ветви представляют собой лингвистические переменные. Выбор критерия для каждого узла

осуществляется на основе алгоритмов, таких как информационный выигрыш или индекс Джини, модифицированных для работы с нечёткими данными. Затем дерево решений обучается на основе исторических данных, благодаря чему определяются оптимальные параметры нечётких множеств и веса критериев. После обучения дерево решений может быть использовано для оценки новых кандидатов. Для этого необходимо предоставить данные о кандидате по каждому из критериев, после чего дерево решений автоматически определит его степень соответствия требованиям вакантной должности.

В конечном счёте, метод нечётких деревьев представляет собой перспективный инструмент для оценки кандидатов на должности в ИТ, который имеет значительные преимущества и пути развития. Например, данный метод хорошо интегрируется с нейронными сетями. Благодаря использованию нейронных сетей работу метода можно в значительной степени оптимизировать. Также данный метод способен на широкий охват по критериям, и поэтому его можно использовать для полноценной, комплексной оценки кандидатов, и вместе с этим учитывать внешние факторы (например, изменение требований рынка ИТ). Метод нечётких деревьев решает задачу повышения объективности и эффективности процесса отбора, учитывая неопределённость и субъективность экспертных оценок, моделируя сложные взаимосвязи между различными критериями оценки.

2.2 Метод анализа иерархий

Метод анализа иерархий (МАИ) представляет собой мощный инструмент поддержки принятия решений, позволяющий декомпозировать сложную задачу на иерархическую структуру, состоящую из цели, критериев и альтернатив, и оценивать их относительную важность посредством парных сравнений. В контексте оценки кандидата на должность в ИТ, МАИ предоставляет возможность структурировать процесс оценки, учесть различные факторы, определяющие успех кандидата, и получить интегральную оценку его соответствия требованиям должности [13].

Структура иерархии для оценки кандидата.

Для применения МАИ к оценке кандидата на должность в ИТ необходимо построить иерархическую структуру, отражающую ключевые аспекты, определяющие успешность кандидата. Типичная структура может включать следующие уровни:

- Цель: выбор наиболее подходящего кандидата на должность;
- Критерии: набор факторов, определяющих соответствие кандидата требованиям должности. Ключевые критерии могут включать:
 - Технические навыки: знание и опыт работы с необходимыми языками программирования, платформами, инструментами и технологиями (например, Java, Python, AWS, SQL).
 - Аналитические способности: умение анализировать сложные проблемы, выявлять ключевые факторы и разрабатывать эффективные решения. Оценивается посредством анализа выполненных проектов, решения задач на логику и критическое мышление.
 - Коммуникативные навыки: способность эффективно общаться с членами команды, заказчиками и другими заинтересованными сторонами. Оценивается в ходе собеседований и групповых дискуссий.
 - Опыт работы: наличие релевантного опыта работы в ИТ-индустрии. Оценивается на основе анализа резюме и рекомендаций.
 - Личностные качества: мотивация, ответственность, обучаемость, умение работать в команде. Оценивается на основе психологических тестов и поведенческих интервью.
- Альтернативы: кандидаты, претендующие на должность.

Процесс оценки с использованием МАИ.

После построения иерархии проводится процесс парных сравнений, в ходе которого эксперты (например, члены комиссии по отбору, технические специалисты, руководители отделов) оценивают относительную важность критериев и альтернатив по отношению друг к другу. Для оценки используется шкала Саати, обычно от 1 до 9, где 1 означает равную важность, а 9 – абсолютное превосходство одного элемента над другим. Результаты парных сравнений представляются в виде матриц, из которых вычисляются векторы приоритетов, отражающие относительную важность каждого критерия и каждой альтернативы. Для обеспечения согласованности оценок проводится проверка коэффициента согласованности, который должен быть меньше 0.1. В противном случае необходимо пересмотреть оценки и повторить процесс вычислений. После получения векторов приоритетов для каждого уровня иерархии проводится агрегирование результатов, позволяющее получить интегральную оценку для каждой альтернативы. Альтернатива с наивысшей интегральной оценкой считается наиболее предпочтительной.

Преимущества применения МАИ в оценке кандидатов:

- Структурированный и систематизированный подход: МАИ позволяет формализовать процесс оценки, выделить ключевые критерии и оценить их относительную важность.
- Учёт различных факторов: МАИ позволяет учитывать как количественные (например, опыт работы, наличие сертификатов), так и качественные (например, личностные качества, мотивация) факторы.
- Объективность и прозрачность: использование парных сравнений и математических расчётов снижает субъективность оценок и повышает прозрачность процесса принятия решений.

- Возможность учёта мнения экспертов: МАИ позволяет привлечь к оценке различных экспертов и учесть их мнения при принятии решений.
- Улучшение качества принимаемых решений: Применение МАИ позволяет принимать более обоснованные и взвешенные решения при выборе кандидатов.

Метод анализа иерархий является эффективным инструментом для оценки кандидатов на должность в ИТ, позволяющим структурировать процесс оценки, учесть различные факторы, определяющие успех кандидата, и получить интегральную оценку его соответствия требованиям должности. Внедрение МАИ в практику отбора персонала позволяет повысить качество принимаемых решений и обеспечить организацию квалифицированными специалистами, способными успешно решать поставленные задачи.

2.3 Метод бинарного выбора

Модель бинарного выбора, как следует из названия, предназначена для анализа ситуаций, где исход имеет два возможных значения. По сути, она отвечает на вопрос: «Что влияет на вероятность выбора одного из двух вариантов?». Зависимая переменная в этой модели бинарна, принимая значения 0 или 1, отражая, состоялось ли событие или нет. Предикторы, или независимые переменные, могут быть как количественными, так и качественными и используются для оценки вероятности того или иного исхода.

Оценка качества модели бинарного выбора – критически важный этап. Классификационная таблица, предложенная Цыплаковым А.А. [14], позволяет оценить, насколько хорошо модель классифицирует наблюдения по их априорным категориям. ROC-кривая, визуализирующая зависимость между долей истинно положительных и ложноположительных исходов, является ещё одним эффективным инструментом для оценки и сравнения моделей. Чем ближе кривая к верхнему левому углу, тем выше предсказательная способность модели.

Логит- и пробит-модели представляют собой два распространённых варианта реализации модели бинарного выбора. Логит-модель, основанная на логистической функции, предполагает, что зависимая переменная подчиняется логистическому распределению. Пробит-модель, в свою очередь, предполагает нормальное распределение зависимой переменной. Обе модели используют метод максимального правдоподобия для оценки параметров и позволяют определить значимость предикторов в определении вероятности наступления события.

Рассмотрим применение модели бинарного выбора в контексте оценки кандидатов на должность в IT-сфере. Предположим, компания использует модель для прогнозирования вероятности успешного прохождения кандидатом испытательного срока (успех = 1, неудача = 0). В качестве предикторов могут выступать: результаты технических тестов (количественный показатель), опыт работы в релевантной области (количественный показатель), оценка на собеседовании (количественный показатель), наличие сертификатов (бинарный признак: есть = 1, нет = 0) и рекомендации с предыдущих мест работы (качественный признак, который можно преобразовать в количественный с помощью шкалы оценок).

Используя исторические данные о предыдущих кандидатах и их результатах на испытательном сроке, компания может построить логит- или пробит-модель. Анализ коэффициентов модели позволит определить, какие факторы наиболее сильно влияют на вероятность успешного прохождения испытательного срока. Например, может оказаться, что результаты технических тестов и опыт работы имеют наибольший вес. Это позволит HR-отделу компании более эффективно отбирать кандидатов, фокусируясь на наиболее значимых критериях и повышая вероятность успешного найма. ROC-кривая и матрица классификации помогут оценить, насколько хорошо модель предсказывает успех или неудачу кандидатов, и принять решение о целесообразности её использования в процессе отбора.

2.4 Сравнительный анализ методов

Проведение технического тестирования является неотъемлемой частью процесса подбора кандидатов на должности в IT-сфере. Эффективная оценка результатов тестирования критически важна для принятия обоснованных решений о найме. Для анализа результатов технического тестирования могут быть использованы различные методы, каждый из которых обладает своими преимуществами и недостатками. В текущей главе мы сравним три метода: метод нечётких деревьев, метод анализа иерархий (МАИ) и метод бинарного выбора, с точки зрения их применимости к задаче оценки результатов технического тестирования.

С помощью метода нечётких деревьев можно моделировать сложные системы с учётом неопределённости и нечёткости. В контексте оценки результатов технического тестирования, данный метод может быть использован для представления логической взаимосвязи между различными критериями оценки (например, знание языка программирования, умение решать алгоритмические задачи, понимание принципов проектирования баз данных) и общим уровнем квалификации кандидата. Ключевой особенностью метода является использование нечёткой логики для обработки неточных и неоднозначных данных, что особенно актуально при оценке качественных аспектов. Главным недостатком метода можно считать субъективность, возникающую из-за необходимости работать с нечёткими данными. Необходимость работы с нечёткими множествами значительно повышает субъективность расчётов. А также, по сравнению с другими методами, анализ результатов с применением нечётких деревьев имеет наибольшую вычислительную сложность и требует использования специализированных инструментов, что ограничивает вариативность и гибкость метода.

В контексте оценки результатов тестирования, метод анализа иерархий может быть использован для определения относительной важности различных критериев оценки и для ранжирования кандидатов на основе их

производительности по этим критериям. Метод предполагает построение иерархии, где на верхнем уровне находится цель (например, выбор лучшего кандидата), на среднем уровне - критерии оценки, а на нижнем уровне - альтернативы (кандидаты). Парные сравнения позволяют определить веса критериев и оценить производительность каждого кандидата по каждому критерию. Затем, с использованием математических расчётов, определяется общий рейтинг каждого кандидата. Самым существенным недостатком метода можно считать его трудоёмкость, возникающую в процессе попарного сравнения критериев. Однако это компенсируется за счёт использования средств программного обеспечения.

В рамках оценки кандидатов, применение пробит или логит модели начинается с определения критериев оценки (предикторов), таких как баллы за тесты по знанию языков программирования, алгоритмов, баз данных, системного администрирования и других релевантных компетенций. Затем формируется обучающая выборка, состоящая из данных о предыдущих кандидатах, где каждому кандидату присваивается метка "подходит" или "не подходит" на основании комплексной оценки, включающей не только результаты тестирования, но и собеседования, опыт работы и другие факторы. Анализ коэффициентов модели позволяет выявить наиболее важные компетенции, определяющие успешность кандидата, что может быть использовано для оптимизации процесса обучения и развития персонала. Однако ключевым недостатком метода можно назвать требование к большому объёму данных для получения надёжных оценок. Недостаток данных может привести к переобучению, когда модель хорошо описывает обучающую выборку, но плохо обобщает на новые данные.

В ходе сравнительного анализа мы сопоставили особенности работы трёх подобных методов, выяснили их основные плюсы и минусы относительно друг друга. Для задачи анализа результатов общего тестирования кандидатов на должность в IT-сфере с учётом различных критериев больше всего подошёл

метод анализа иерархий. Данный метод подходит для справедливой оценки каждого кандидата, его можно использовать гибко, даже при сравнении критериев разной степени значимости. Разработав информационную систему для проведения тестирования с последующим анализом результатов методом анализа иерархий, можно получить эффективный инструмент для оценки большого числа кандидатов на должность с разработкой рекомендаций в вопросе выбора наиболее подходящего человека.

3 Разработка информационной системы подбора и оценки кандидатов на должность в IT-индустрии

3.1 Разработка базы данных

Для эффективного хранения и организации информации о кандидатах, была разработана база данных с использованием PostgreSQL. Система управления базами данных PostgreSQL на данный момент является одной из самых продвинутых и надёжных, обеспечивая гарантию работоспособности даже в условиях высокой нагрузки.

В ходе выполнения работы было составлено 12 таблиц: таблица «Кандидаты», таблица - «Рекрутеры», пять таблиц - с тестами и пять таблиц с ответами кандидатов на тесты (рисунок 1).

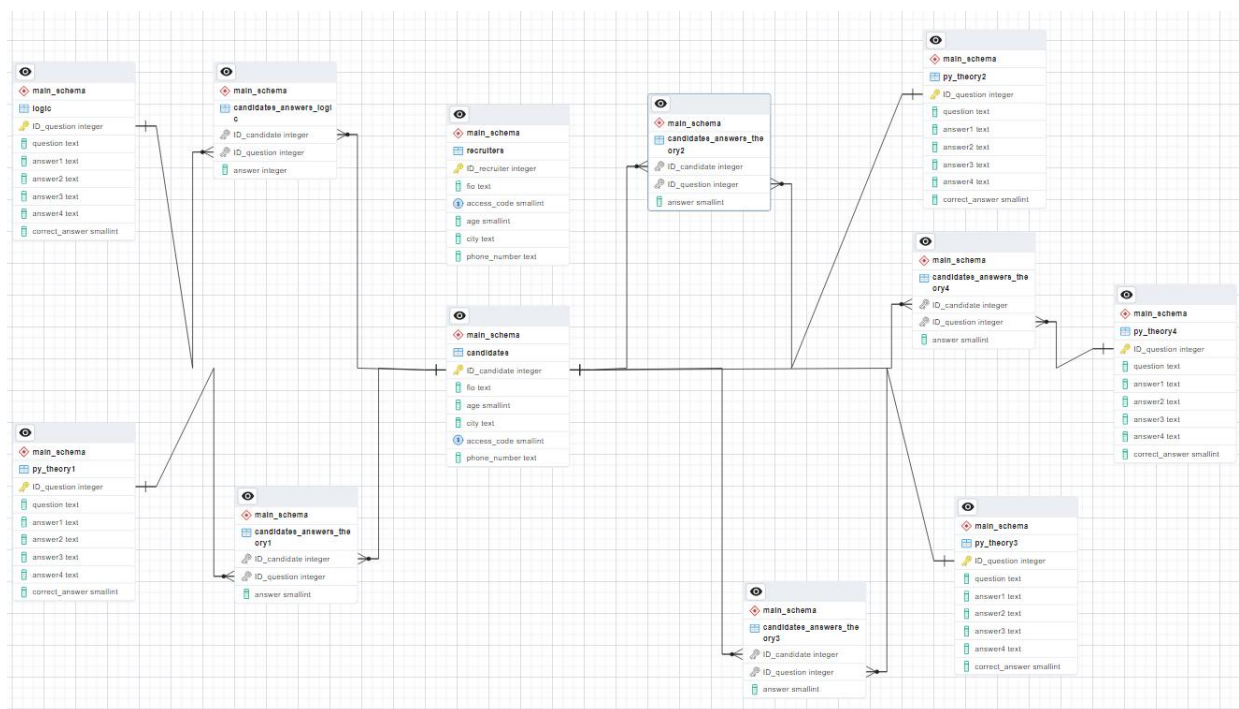


Рисунок 1 - Схема базы данных

Подробное описание:

Таблица «candidates». Содержит информацию о зарегистрированных в системе кандидатах. Каждый кандидат имеет свой уникальный ID, ФИО,

возраст, город, код доступа, номер телефона. Пример с тестовыми данными на рисунке 2.

| | ID_candidate [PK] integer | fio text | age smallint | city text | access_code smallint | phone_number text |
|---|------------------------------|--------------------------------|-----------------|-----------------|-------------------------|----------------------|
| 1 | 1 | Никольский Платон Алексеевич | 19 | Санкт-Петербург | 1234 | 89134562365 |
| 2 | 2 | Тихонова Василиса Данииловна | 23 | Санкт-Петербург | 1235 | 89511234567 |
| 3 | 3 | Михайлова Алиса Александровна | 34 | Санкт-Петербург | 1236 | 891198777654 |
| 4 | 4 | Михайлова Александра Фёдоровна | 42 | Москва | 1237 | 89997655675 |
| 5 | 5 | Воробьев Андрей Вячеславович | 26 | Санкт-Петербург | 1238 | 89112233322 |
| 6 | 6 | Тихомиров Артём Григорьевич | 28 | Санкт-Петербург | 1239 | 89315678790 |

Рисунок 2 - Тестовые данные для таблицы «candidates»

Таблица «recruiters». Содержит информацию о зарегистрированных в системе людей, занимающихся наймом. Каждый рекрутер имеет свой уникальный ID, ФИО, возраст, город, код доступа, номер телефона. Пример с тестовыми данными на рисунке 3.

| | ID_recruiter [PK] integer | fio text | access_code smallint | age smallint | city text | phone_number text |
|---|------------------------------|---------------------------|-------------------------|-----------------|-----------------|----------------------|
| 1 | 1 | Кулагин Филипп Леонидович | 2345 | 35 | Санкт-Петербург | 89119752823 |

Рисунок 3 - Тестовые данные для таблицы «recruiters»

Таблицы ru_theory1, ru_theory2, ru_theory3, ru_theory4, logic имеют одинаковую структуру: ID вопроса, сам вопрос, четыре варианта ответа и номер правильного ответа. Пример с тестовыми данными таблицы ru_theory1 на рисунке 4. Примеры с тестовыми данными для остальных таблиц в приложении 1.

| | ID_question [PK] integer | question text | answer1 text | answer2 text | answer3 text | answer4 text | correct_answ smallint |
|----|-----------------------------|------------------|-----------------|-----------------|-----------------------------------|-------------------------------------|--------------------------|
| 1 | 1 | Что тако... | Тип пере... | Тип пере... | Тип переменной определяет... | Тип переменной определяется в... | 2 |
| 2 | 2 | Разница ... | Список (l... | Список (l... | Список (list) - содержит в себ... | Список (list) - последовательнос... | 1 |
| 3 | 3 | Что тако... | Генерато... | Генерато... | Генератор - часть компилят... | Генератор – это специальный ти... | 1 |
| 4 | 4 | Что тако... | Декорат... | Декорат... | Декораторы – это специаль... | Декораторы – это функции, позв... | 4 |
| 5 | 5 | Что тако... | Множест... | Множест... | Множественное наследован... | Множественное наследование – ... | 4 |
| 6 | 6 | Объясни... | == сравн... | == сравн... | == сравнивает ID объектов в ... | == сравнивает указатели на объе... | 2 |
| 7 | 7 | Что тако... | GIL – это ... | GIL – это ... | GIL – это механизм в CPython... | GIL – это инструмент отладки мн... | 3 |
| 8 | 8 | Назначе... | Специал... | Позволя... | Встроенные библиотеки Pyth... | Инструменты для компиляции P... | 2 |
| 9 | 9 | Как обра... | Используй... | Исключе... | Исключения в Python автом... | Для обработки исключений испо... | 1 |
| 10 | 10 | Разница ... | Позицио... | Позицио... | Позиционные аргументы пе... | Позиционные аргументы - это гл... | 3 |

Рисунок 4 - Содержание первого теста в таблице «py_theory1»

Таблицы candidates_answers_theory1, candidates_answers_theory2, candidates_answers_theory3, candidates_answers_theory4, candidates_answers_logic имеют одинаковую структуру: ID кандидата (отвечавшего на вопрос), ID вопроса и выбранный кандидатом вариант ответа. Пример с тестовыми данными таблицы candidates_answers_theory1 на рисунке 5. Примеры с тестовыми данными для остальных таблиц в приложении 1.

| | ID_candidate integer | ID_question integer | answer smallint |
|----|-------------------------|------------------------|--------------------|
| 1 | 1 | 1 | 2 |
| 2 | 1 | 2 | 1 |
| 3 | 1 | 3 | 1 |
| 4 | 1 | 4 | 4 |
| 5 | 1 | 5 | 4 |
| 6 | 1 | 6 | 2 |
| 7 | 1 | 7 | 3 |
| 8 | 1 | 8 | 2 |
| 9 | 1 | 9 | 1 |
| 10 | 1 | 10 | 3 |
| 11 | 2 | 1 | 2 |
| 12 | 2 | 2 | 1 |
| 13 | 2 | 3 | 2 |
| 14 | 2 | 4 | 2 |
| 15 | 2 | 5 | 2 |
| 16 | 2 | 6 | 4 |
| 17 | 2 | 7 | 4 |
| 18 | 2 | 8 | 4 |
| 19 | 2 | 9 | 4 |
| 20 | 2 | 10 | 4 |
| 21 | 3 | 1 | 2 |

Рисунок 5 - Пример записей в таблице «candidates_answers_theory1»

3.2 Применяемые тесты для проведения оценки кандидатов

Тестирование кандидатов на должность Python-разработчика охватывает широкий спектр тем, от базовых знаний языка до более сложных концепций и практических навыков. При составлении тестов была поставлена задача проверить знание основных принципов работы языка и необходимые сопутствующие знания. Каждый тест состоит из десяти вопросов на указанную тему. Всего за полностью правильно пройденный тест можно получить десять баллов. Соответственно, за каждый правильный ответ кандидат получает один балл.

Первый тест содержит вопросы, затрагивающие базовые знания принципов языка программирования Python и его особенностей. Ниже приведено два тестовых вопроса:

Вопрос 1: Что такое динамическая типизация? Её преимущества.

Варианты ответа:

- Тип переменной определяется при компиляции. Преимущество: «избегание» ошибок типизации.
- Тип переменной определяется во время выполнения программы. Преимущество: возможность переназначить переменной значения разных типов.
- Тип переменной определяется при компиляции. Преимущество: скорость разработки.
- Тип переменной определяется во время выполнения программы. Преимущество: увеличивается скорость выполнения программы.

Вопрос 8: Назначение list comprehensions и dictionary comprehensions.

Варианты ответа:

- Специальные функции для работы с файлами, позволяющие считывать данные в список или словарь. Они оптимизированы для работы с большими объемами данных.

- Позволяют создавать новые списки и словари, применяя выражения к элементам существующих итерируемых объектов в более компактной и читаемой форме, чем традиционные циклы `for`.

- Встроенные библиотеки Python для создания графических интерфейсов. List comprehensions используются для создания списков виджетов, а dictionary comprehensions – для организации этих виджетов в иерархическую структуру.

- Инструменты для компиляции Python-кода в машинный код. List comprehensions оптимизирует циклы для работы со списками, а dictionary comprehensions оптимизирует операции со словарями.

Остальные вопросы первого теста вложены в приложение 2.

Второй тест содержит вопросы на знание основных библиотек языка Python. Владение самыми распространёнными библиотеками поможет гибко подходить к решению трудных задач. Ниже приведено два тестовых вопроса:

Вопрос 1: Выберите правильное назначение модуля `os`.

Варианты ответов:

- Модуль `os` предназначен для работы с базами данных. Он обеспечивает интерфейс для подключения к различным СУБД, выполнения SQL-запросов и управления данными.

- Модуль `os` используется для создания графического интерфейса пользователя. Он содержит классы и функции для создания окон, кнопок, текстовых полей и других элементов управления.

- Модуль `os` предоставляет функции для взаимодействия с операционной системой. Позволяет выполнять операции, зависящие от используемой операционной системы.

- Модуль `os` необходим для выполнения математических расчётов повышенной точности. Он реализует сложные алгоритмы для решения математических задач.

Вопрос 8: Какова роль модуля `math`? Как вычислить логарифм числа?

Варианты ответов:

- Модуль `math` предоставляет математические функции и константы для выполнения различных математических операций. Для вычисления логарифма числа используется функция `math.log()`.

- Модуль `math` содержит базовые арифметические операции, такие как сложение и вычитание. Для вычисления логарифма числа используется функция `math.ln()`.

- Модуль `math` используется для работы со строками и текстом. Для вычисления логарифма числа используется функция `math.logarithm()`.

- Модуль `math` позволяет работать с комплексными числами и матрицами. Для вычисления логарифма числа используется функция `math.logb()`.

Остальные вопросы второго теста вложены в приложение 2.

Третий тест содержит вопросы на знания управления проектами, разрешения зависимостей, использования виртуальных окружений. Данный тест нужен для оценки понимания основных концепций и инструментов работы с проектами и выявить способности применять правильные инструменты для решения конкретных задач. Ниже приведено два тестовых вопроса:

Вопрос 1: Какой менеджер пакетов является стандартным и наиболее распространенным в Python?

Варианты ответов:

- conda.
- poetry.
- pip.
- art.

Вопрос 8: Какое преимущество предоставляет использование Docker для переноса проектов с Python?

Варианты ответов:

- Упрощает создание виртуальных окружений.
- Гарантирует максимальную изоляцию и воспроизводимость среды выполнения.
- Автоматически управляет зависимостями операционной системы.
- Ускоряет процесс установки пакетов.

Остальные вопросы третьего теста вложены в приложение 2.

Четвёртый тест содержит вопросы на тему асинхронного программирования с использованием библиотеки «`asyncio`». Знание библиотеки и умение с ней работать может существенно повлиять на работу с программами, требующими особой эффективности и отзывчивости. Данная библиотека используется для написания кода, который одновременно выполняет множество операций без использования многопоточности. Ниже приведено два тестовых вопроса:

Вопрос 1: Какое ключевое слово используется для определения корутины в Python asyncio?

- `def`.
- `async`.
- `thread`.
- `process`.

Вопрос 8: Какую функцию выполняет `asyncio.run(main())`?

- Создает новый поток для выполнения корутины `main()`.
- Запускает уже существующий `event loop` и регистрирует в нем `main()`.
- Создает новый `event loop`, запускает корутину `main()` и закрывает цикл после завершения.
- Компилирует корутину `main()` в машинный код.

Остальные вопросы четвёртого теста вложены в приложение 2.

Пятый тест содержит вопросы, проверяющие способности кандидата анализировать информацию. Для решения тестовых заданий требуются базовые математические знания и умения. Ниже приведено два тестовых вопроса:

Вопрос 1: Дана последовательность чисел: 1, 8, 27, 64, 125... Какое число будет стоять на десятом месте?

- 1000.
- 1200.
- 725.

- 975.

Вопрос 8: В школе 800 учеников, из них 30% — ученики начальной школы. Среди учеников средней и старшей школы 20% изучают немецкий язык. Сколько учеников в школе изучают немецкий язык, если в начальной школе немецкий язык не изучается?

- 112.
- 114.
- 116.
- 118.

Остальные вопросы пятого теста вложены в приложение 2.

3.3 Программная реализация приложения

Для проведения тестирования, а также для взаимодействия с базой данных необходимо разработать программу, основным функционалом которой является обеспечение доступа к БД по запросу данных (запросы на получение, изменение, запись, удаление); взаимодействие с пользователем: в нашем случае пользователями могут быть тестируемый и рекрутер; проведение анализа полученных результатов тестов методом анализа иерархий. Для написания программы был выбран язык программирования Python, поскольку данный язык является одним из самых востребованных на сегодняшний день, в том числе за счёт универсальности в вопросе возможности решения широкого спектра задач. Средств языка Python для воссоздания необходимого функционала будет достаточно.

Конечный проект содержит шесть основных файлов: «main.py», «menu.py», «menu_recruiter.py», «analys.py», «priorities.py» «server.py». Рассмотрим подробнее каждый из них.

Файл «server.py»

Основное назначение: обеспечение взаимодействия с базой данных. В файле реализованы все необходимые функции получения, изменения, добавления и удаления записей, используемые в работе основной части кода.

Для взаимодействия с БД используется библиотека «psycopg2». Подключение и начало работы с БД:

```
import psycopg2
connection = psycopg2.connect(user="postgres",
                              password="password1",
                              host="127.0.0.1",
                              port="5432",
                              database="candidate_testing_db")
cursor = connection.cursor()
cursor.execute('SET search_path TO main_schema,public')
```

Для добавления нового кандидата в таблицу используется функция «add_candidate»:

```
def add_candidate(fio, age, city, access_code, phone_number):
    try:
        # Находим максимальный существующий номер кандидата и увеличиваем на 1
        cursor.execute('SELECT MAX("ID_candidate") FROM candidates')
        max_number = cursor.fetchone()[0]
        new_number = 1 if max_number is None else max_number + 1

        # SQL запрос для вставки новой записи
        insert_query = """
            INSERT INTO candidates ("ID_candidate", fio, age, city, access_code, phone_number)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        record_to_insert = (new_number, fio, age, city, access_code, phone_number)

        # Выполняем запрос
        cursor.execute(insert_query, record_to_insert)
        connection.commit()
        return True, access_code

    except (Exception, psycopg2.Error) as error:
        connection.rollback()
        return False, str(error)
```

Для добавления нового рекрутера в таблицу используется функция «add_recruiter», работающая аналогичным образом.

Для подсчёта количества кандидатов используется функция «get_candidates_count»:

```
def get_candidates_count():
    try:
        # SQL запрос для подсчета всех записей в таблице candidates
        select_query = """
            SELECT COUNT(*)
            FROM candidates
        """

        # Выполняем запрос
        cursor.execute(select_query)
        result = cursor.fetchone()

        # Возвращаем количество записей
        if result:
            return True, result[0]
        else:
            return False, "Не удалось получить количество кандидатов"
```



```
except (Exception, psycopg2.Error) as error:
    return False, str(error)
```

Для подсчёта количества рекрутеров используется функция «get_recruitesr_count». Она работает аналогичным образом.

Для удаления кандидата используется функция «delete_candidate_by_id»:

```
def delete_candidate_by_id(id_candidate):
    try:
        # SQL запросы для удаления ответов кандидата из всех таблиц с ответами
        delete_answers_queries = [
            """DELETE FROM candidates_answers_theory1 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory2 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory3 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory4 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_logic WHERE "ID_candidate" = %s"""
        ]

        # Удаляем ответы кандидата из всех таблиц
        for query in delete_answers_queries:
            cursor.execute(query, (id_candidate,))

        # SQL запрос для удаления кандидата по ID
        delete_query = """
            DELETE FROM candidates
            WHERE "ID_candidate" = %s
            """

        # Выполняем запрос на удаление кандидата
        cursor.execute(delete_query, (id_candidate,))

        # Подтверждаем изменения
        connection.commit()

        # Проверяем, была ли удалена запись
        if cursor.rowcount > 0:
            return True, "Кандидат успешно удален"
        else:
            return False, "Кандидат с таким ID не найден"

    except (Exception, psycopg2.Error) as error:
        # Откатываем изменения в случае ошибки
        connection.rollback()
        return False, str(error)
```

Внутри данной функции, помимо непосредственного удаления информации о тестируемом из таблицы «candidates», реализовано удаление записей из других таблиц с информацией о пройденных тестах во избежание

ошибок. Поскольку таблица «recruiters» в базе является обособленной (не имеет связей), функция удаления записи о нанимателе несколько проще:

```
def delete_recruiter_by_id(id_recruiter):
    try:
        # SQL запрос для удаления рекрутера по ID
        delete_query = """
            DELETE FROM recruiters
            WHERE "ID_recruiter" = %s
        """

        # Выполняем запрос на удаление рекрутера
        cursor.execute(delete_query, (id_recruiter,))

        # Подтверждаем изменения
        connection.commit()

        # Проверяем, была ли удалена запись
        if cursor.rowcount > 0:
            return True, "Рекрутер успешно удален"
        else:
            return False, "Рекрутер с таким ID не найден"

    except (Exception, psycopg2.Error) as error:
        # Откатываем изменения в случае ошибки
        connection.rollback()
        return False, str(error)
```

Функция получения некоторой информации о кандидате по его ID:

```
def get_candidate_info_by_id(id_candidate):
    try:
        # SQL запрос для поиска информации о кандидате по ID
        select_query = """
            SELECT fio, city, phone_number
            FROM candidates
            WHERE "ID_candidate" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        result = cursor.fetchone()

        # Если найден кандидат, возвращаем его данные
        if result:
            return True, {
                'fio': result[0],
                'city': result[1],
                'number': result[2]
            }
        else:
            return False, "Кандидат с таким ID не найден"
```

```
except (Exception, psycopg2.Error) as error:
    return False, str(error)
```

Для получения некоторой информации о кандидате по его ID используется функция «get_recruitesr_info». Она работает аналогичным образом.

Для получения ответов пользователя по первому тесту используется функция «get_answers_list_by_candidate_theory1»:

```
def get_answers_list_by_candidate_theory1(id_candidate):
    try:
        # SQL запрос для получения ответов кандидата
        select_query = """
            SELECT answer
            FROM candidates_answers_theory1
            WHERE "ID_candidate" = %s
            ORDER BY "ID_question"
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        results = cursor.fetchall()

        # Преобразуем результаты в список чисел
        if results:
            answers = [row[0] for row in results]
            return True, answers
        else:
            return False, "Ответы не найдены"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)
```

Для получения результатов по остальным тестам используются аналогичные функции.

Для выполнения запроса на сохранение ответа пользователя по первому тесту используется функция «save_answer»:

```
def save_answer(id_candidate, id_question, answer):
    try:
        # SQL запрос для вставки ответа кандидата
        insert_query = """
            INSERT INTO candidates_answers_theory1 ("ID_candidate", "ID_question", answer)
            VALUES (%s, %s, %s)
        """

        # Выполняем запрос
```

```
cursor.execute(insert_query, (id_candidate, id_question, answer))
connection.commit()
```

```
return True, "Ответ успешно сохранен"
```

```
except (Exception, psycopg2.Error) as error:
    return False, str(error)
```

Для выполнения запроса на сохранение ответа пользователя на по остальным тестам используется аналогичная функция.

Полноценное содержание файла «server.py» вынесено в приложение 3.

Файл «[nalis.py](#)».

Основное назначение: реализация метода анализа иерархий для оценки кандидатов на основе их результатов тестирования.

Функция «`get_coorect_answers_count_theory1`» подсчитывает количество правильных ответов в первом тесте у кандидата с указанным ID:

```
def get_correct_answers_count_theory1(id_candidate):
    try:
        # Получаем ответы кандидата
        success, candidate_answers = get_answers_list_theory1(id_candidate)
        if not success:
            return False, candidate_answers # Возвращаем ошибку

        # Получаем правильные ответы
        success, correct_answers = get_answers_listt_theory1()
        if not success:
            return False, correct_answers # Возвращаем ошибку

        # Считаем количество правильных ответов
        correct_count = 0
        for candidate_answer, correct_answer in zip(candidate_answers, correct_answers):
            if candidate_answer[1] == correct_answer: # candidate_answer[1] - это ответ кандидата
                correct_count += 1

        return True, correct_count

    except Exception as error:
        return False, str(error)
```

Также есть подобные ей функции «get_coorect_answers_count_theory2», «get_coorect_answers_count_theory3», «get_coorect_answers_count_theory4», «get_coorect_answers_count_logic», работающие аналогичным образом.

Далее используется функция «geometrin_mean» для расчёта среднего геометрического:

```
def geometric_mean(numbers):  
    product = 1  
    for num in numbers:  
        product *= num  
    return product ** (1/len(numbers))
```

Данная функция применяется для расчёта среднего геометрического каждой строки матрицы коэффициентов. Затем используется функция «final_priorities», которая нормирует вектор приоритетов:

```
def final_prioritees(vector):  
    sum_vector = sum(vector)  
    for i in range(len(vector)):  
        vector[i] = vector[i] / sum_vector  
    return vector
```

Финальная функция «get_candidate_score» принимает ID кандидата и возвращает его оценку за тестирование с учётом коэффициентов значимости тестов, используя остальные функции для расчётов:

```
def get_candidate_score(id_candidate):  
    try:  
        print(f"\nРасчет score для кандидата {id_candidate}:")  
  
        priorities = get_priorities()  
        print(f"Полученные приоритеты: {priorities}")  
  
        if isinstance(priorities, tuple) and not priorities[0]:  
            return False, priorities[1]  
  
        vector_priorities = []  
        for row in priorities:  
            mean = geometric_mean(row)  
            print(f"Геометрическое среднее для строки {row}: {mean}")  
            vector_priorities.append(mean)  
  
        vector_priorities = final_prioritees(vector_priorities)  
        print(f"Нормализованные приоритеты: {vector_priorities}")  
  
        # Получаем результаты тестов
```

```

test_results = [
    get_correct_answers_count_theory1(id_candidate),
    get_correct_answers_count_theory2(id_candidate),
    get_correct_answers_count_theory3(id_candidate),
    get_correct_answers_count_theory4(id_candidate),
    get_correct_answers_count_logic(id_candidate)
]
print(f"Результаты тестов: {test_results}")

# Проверяем успешность получения ответов и извлекаем количество правильных
ответов
correct_answers = []
for success, count in test_results:
    if not success:
        print(f"Ошибка при получении результатов теста: {count}")
        return False, count
    correct_answers.append(count)

print(f"Количество правильных ответов: {correct_answers}")

# Вычисляем итоговый score
score = 0
for i in range(5):
    score += vector_priorities[i] * correct_answers[i]
    print(f"Вклад теста {i+1}: {vector_priorities[i]} * {correct_answers[i]} =
{vector_priorities[i] * correct_answers[i]}")

print(f"Итоговый score: {score}")
return True, score
except Exception as error:
    print(f"Ошибка при расчете score: {str(error)}")
    return False, str(error)

```

Файл «main.py».

Основное назначение: запуск программы и вывод окна авторизации пользователей. Для создания оконного приложения была выбрана библиотека PyQt5. Данная библиотека обладает набором виджетов, а также системой сигналов, которые были использованы в работе. Также PyQt5 обладает хорошей производительностью за счёт того, что является обёрткой над Qt, написанной на C++. После подключения библиотек были прописаны общие стили для окна. Глобально файл содержит два класса: WelcomWindow и RegistrationWindow, через которые пользователи получают доступ к программе. Подробный код файла вынесен в приложение 3.

Файл «menu.py».

Основное назначение: проведение тестирования кандидатов. В данном файле также используется библиотека PyQt5 для оконного отображения. Здесь реализовано одиннадцать классов: пять классов для проведения непосредственно самого тестирования по пяти тестам, пять классов для отображения результатов в виде таблицы и основной класс окна с навигацией. Подробный код файла вынесен в приложение 3.

Файл «menu_recruiters».

Основное назначение: отображение результатов тестирования всех кандидатов, а также рекомендация программы после проведения тестирования. В данном файле также используется библиотека PyQt5 для оконного отображения. Здесь реализованы пять классов: один класс для меню с навигацией, второй класс для отображения оценок за тест для каждого кандидата, отдельный класс для добавления нового рекрутера в базу данных, четвёртый класс для изменения таблицы приоритетов тестов и класс, выводящий результаты расчётов МАИ. Подробный код файла вынесен в приложение 3.

Файл «priorities.py»

Основное назначение: хранение и предоставление доступа к изменению значений таблицы приоритетов. В файле содержится переменная «priorities», в которой хранятся значения сравнений тестов по степени их важности. Также прописано две функции: для доступа и записи значений в переменную.

Далее будет описана логика программы на примере использования.

На рисунке 6 показано окно авторизации для пользователей: нанимателей и тестируемых (рисунок 7). При нажатии на кнопку «Регистрация», открывается окно для регистрации нового кандидата. Для успешной регистрации необходимо заполнить указанные данные.

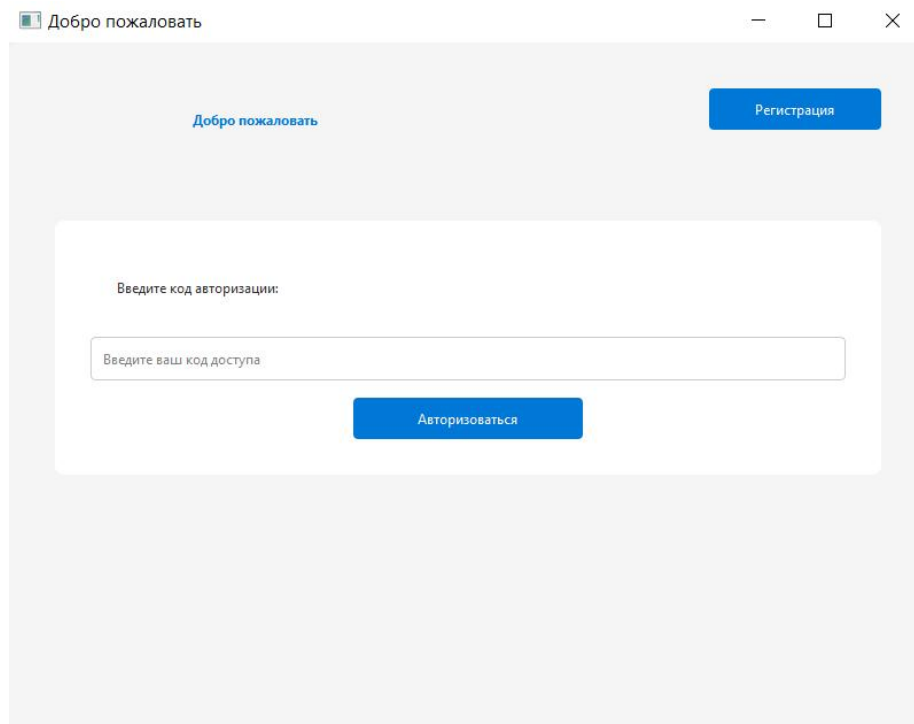


Рисунок 6 - Окно авторизации

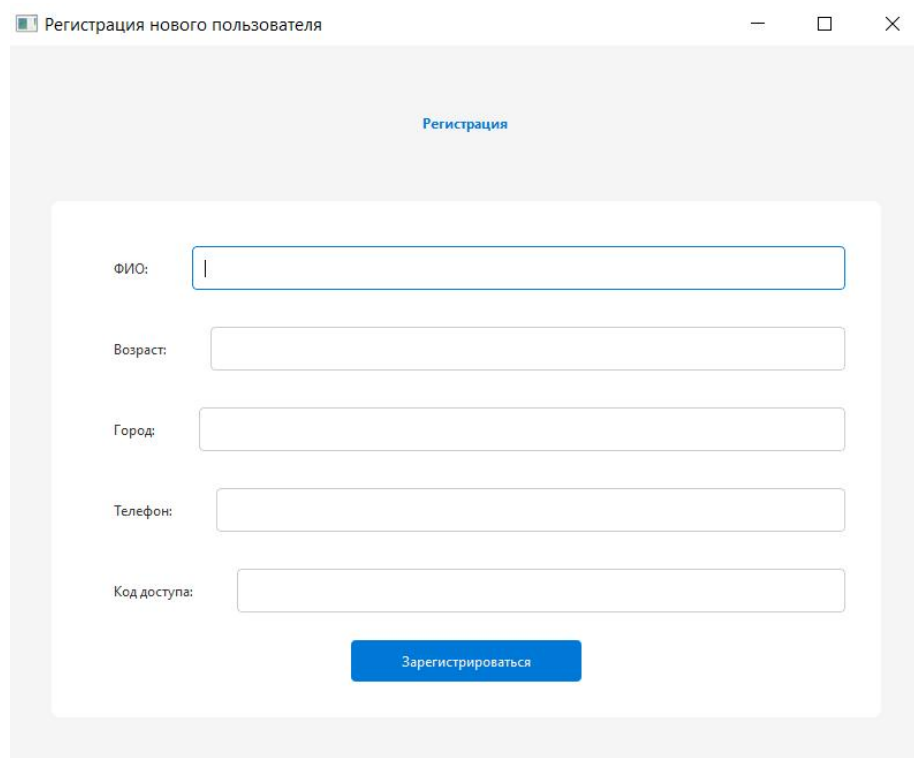


Рисунок 7 - Окно регистрации нового тестируемого

У каждого пользователя, зарегистрированного в системе, есть свой уникальный код доступа. Именно по нему происходит вход в основную

программу. При авторизации под существующим уникальным кодом, система распознаёт его тип (тестируемый или рекрутер) и открывает нужное окно (рисунок 8).

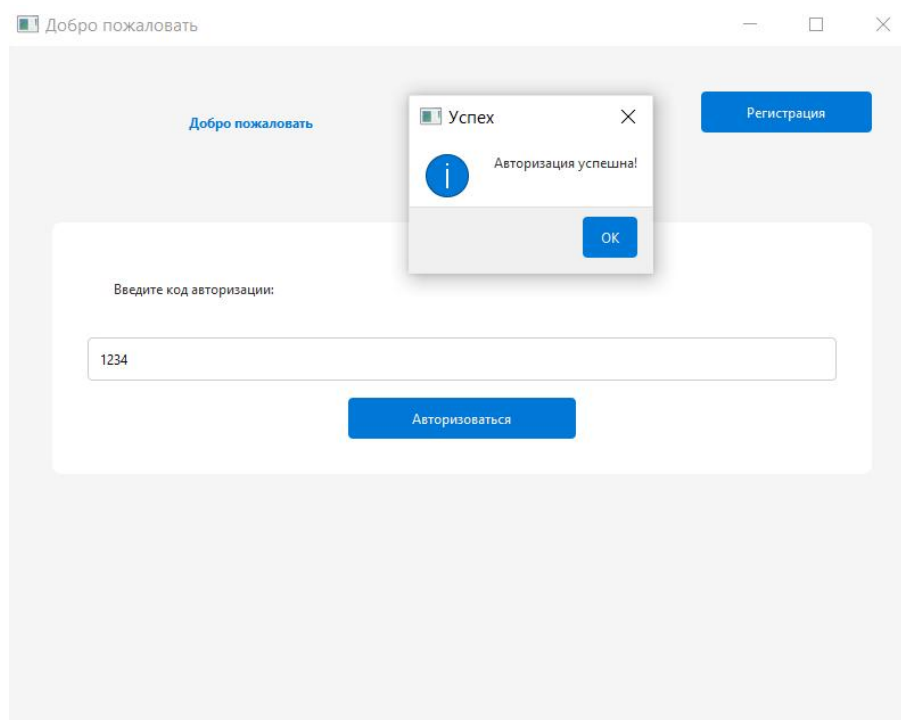


Рисунок 8 - Успешная авторизация

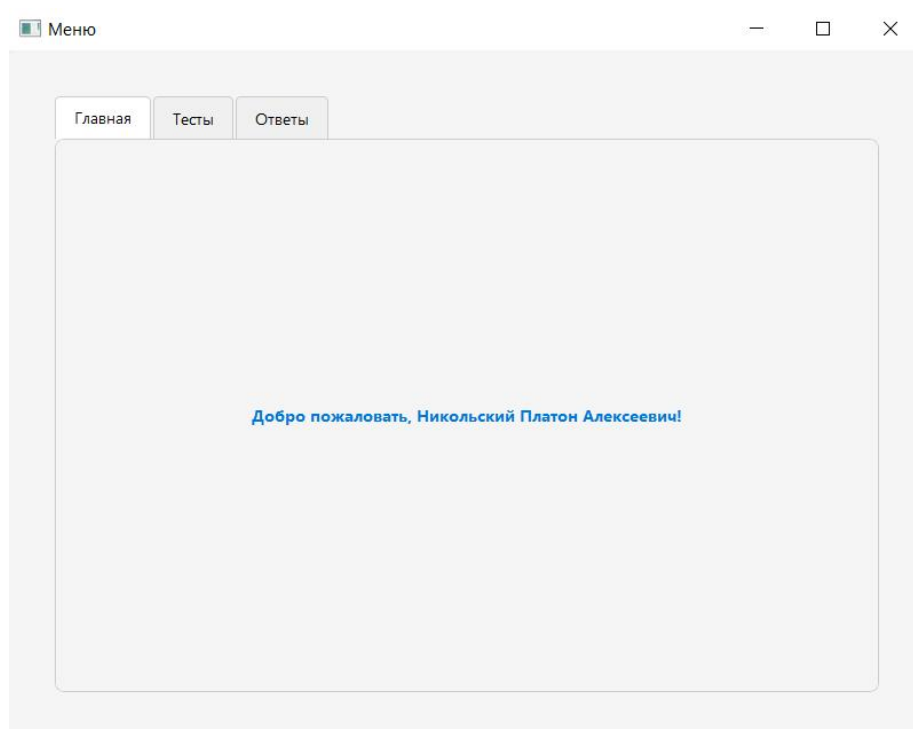


Рисунок 9 - Главное меню кандидата

На рисунке 9 отображена приветственная вкладка, открывающаяся после прохождения авторизации. Далее тестируемый может перейти на вкладку «Тесты» (рисунок 10) или на вкладку «Ответы» (рисунок 11).

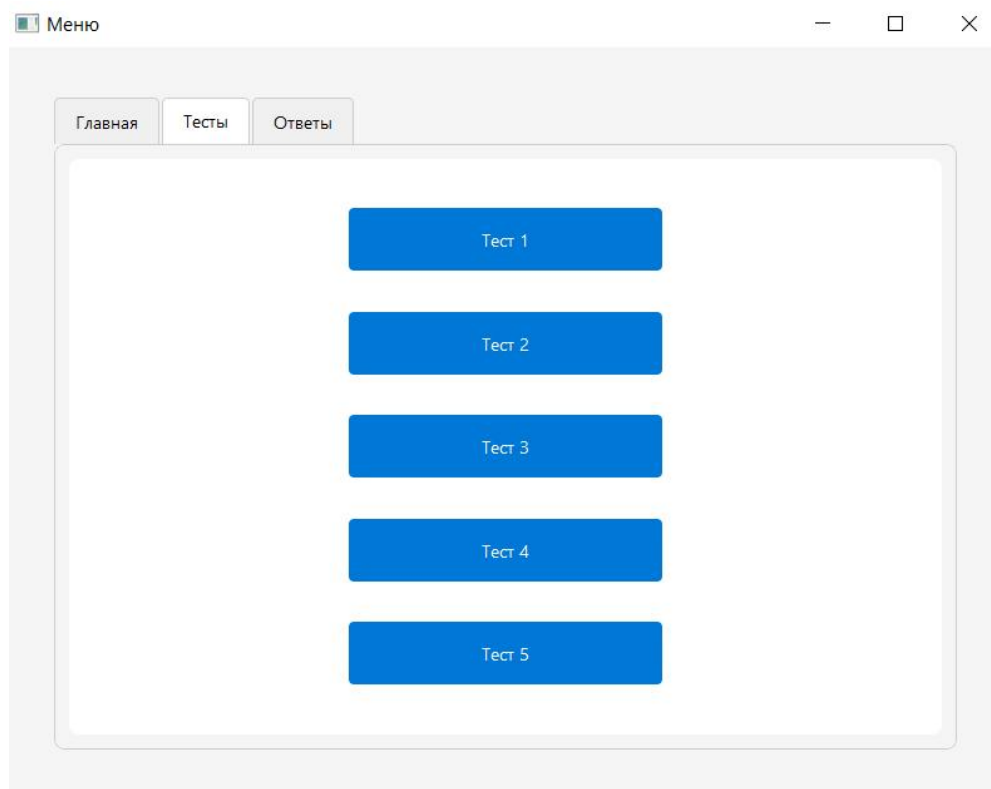


Рисунок 10 - Вкладка «Тесты» в интерфейсе кандидата

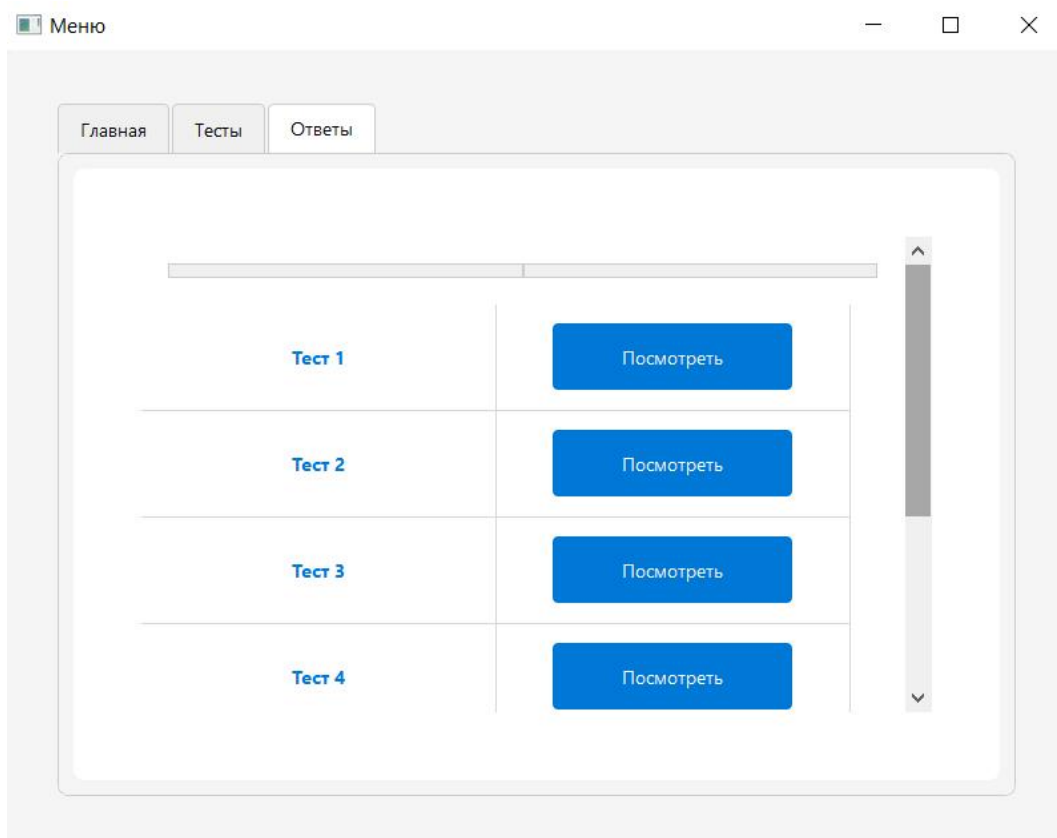


Рисунок 11 - Вкладка «Ответы» в интерфейсе кандидата

При попадании на вкладку «Тесты» кандидатам предложено пройти пять тестов, для каждого из которых своя кнопка. При нажатии на кнопку «Тест 1», кандидат попадает на информационное окно перед началом теста (рисунок 12).

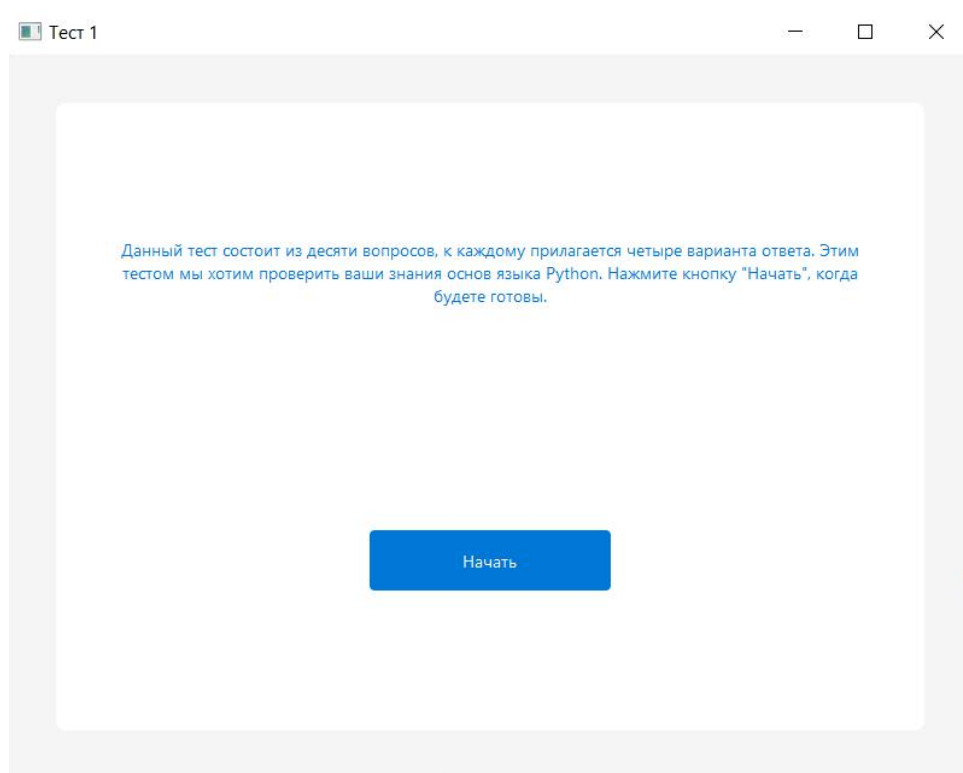


Рисунок 12 - Окно запуска первого теста

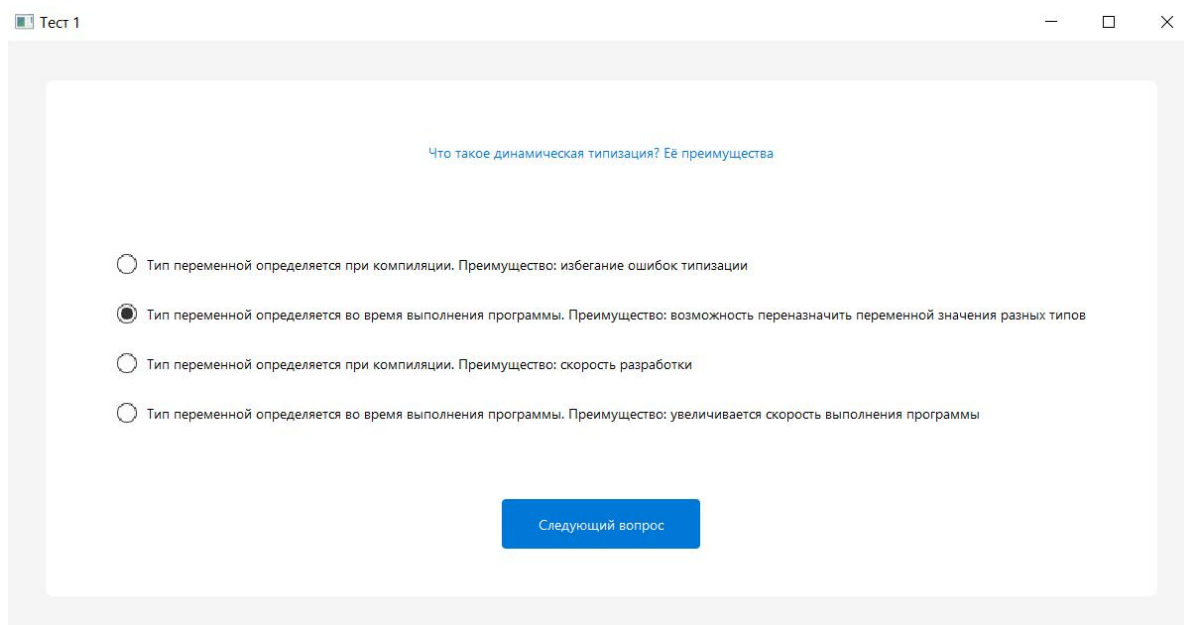


Рисунок 13 - Окно с первым вопросом первого теста

Когда кандидат будет готов проходить тест, он нажимает «Начать» и приступает к решению. На рисунке 13 показано, как кандидат проходит тестирование на примере первого вопроса первого теста. Далее на рисунке 14

отображено окно завершения теста и уведомлением, что результаты записаны в базу данных.

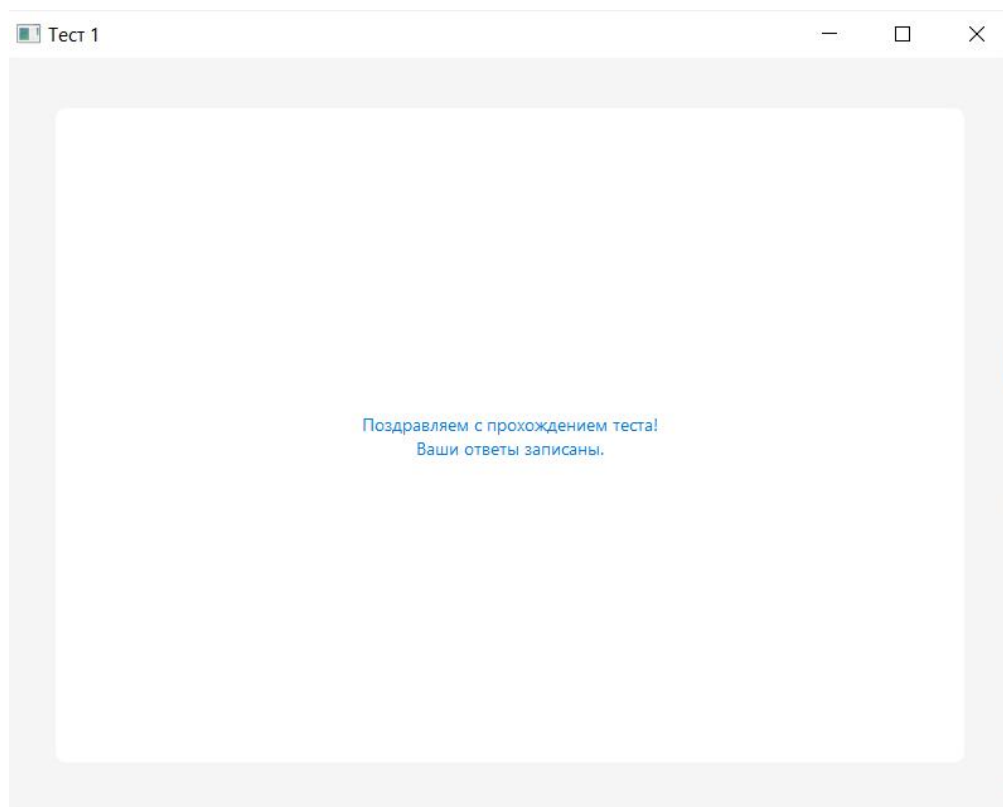


Рисунок 14 - Окно завершения теста

По завершении любого из тестов есть возможность ознакомиться со своими ответами на вопросы ещё раз (рисунок 15). Тем не менее, саму оценку за тест было решено не сообщать и предоставить нанимающей стороне решить, оглашать результат или нет.

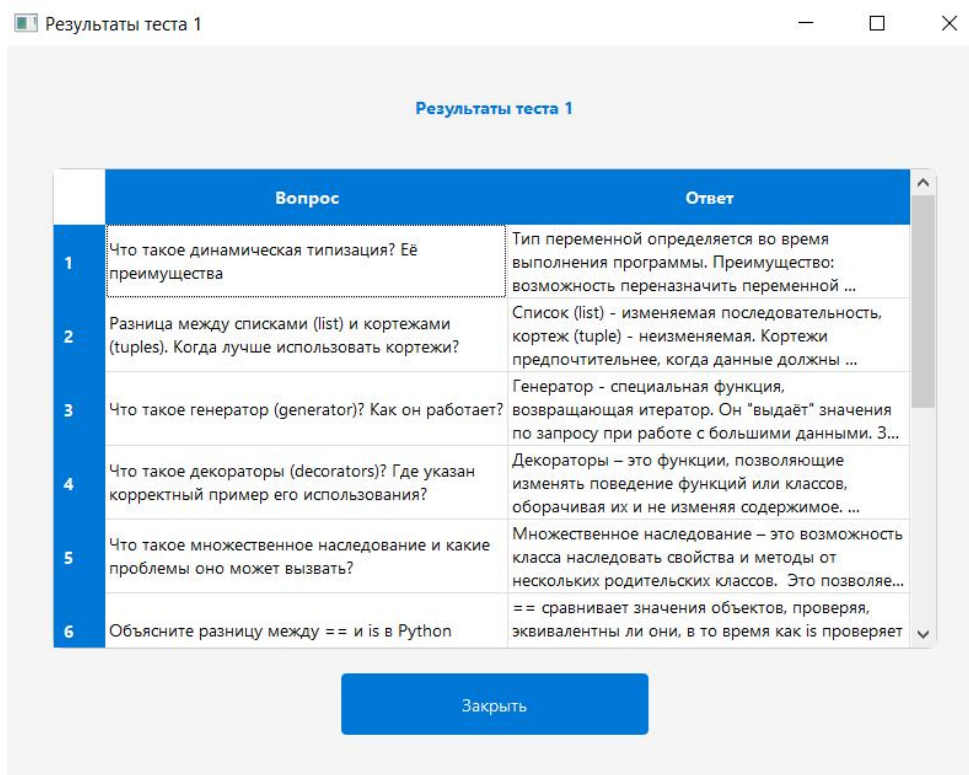


Рисунок 15 - Просмотр ответов кандидата на первый тест

Теперь перейдём к просмотру меню для рекрутеров (рисунок 16). Приветственное окно ничем не отличается от такого же окна у кандидатов. Однако, здесь отображены другие вкладки: «Главная», «Кандидаты», «Пользователи», «Аналитика».

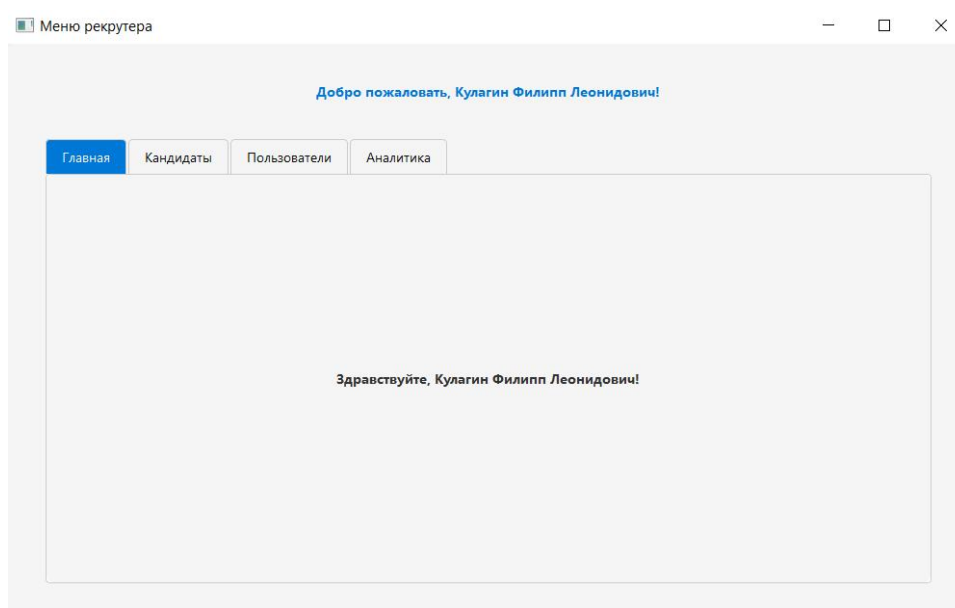


Рисунок 16 - Меню нанимателя

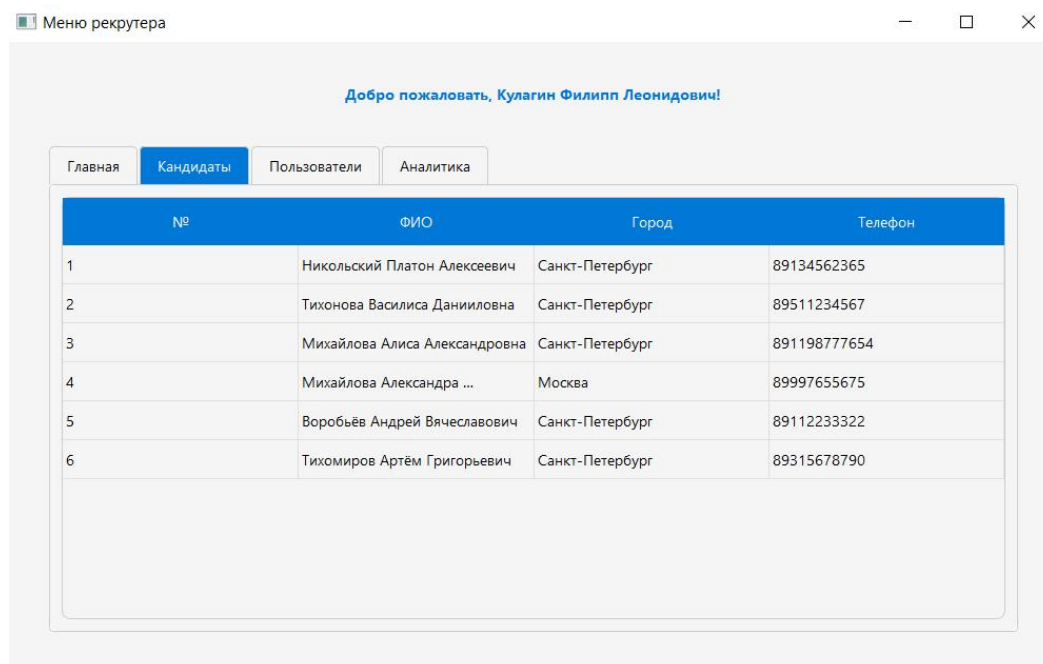


Рисунок 17 - Вкладка «Кандидаты»

На вкладке «Кандидаты» (рисунок 17) отображается таблица с информацией о всех зарегистрированных в базе тестируемых. Нажав на любую ячейку таблицы можно вызвать контекстное меню (рисунок 18).

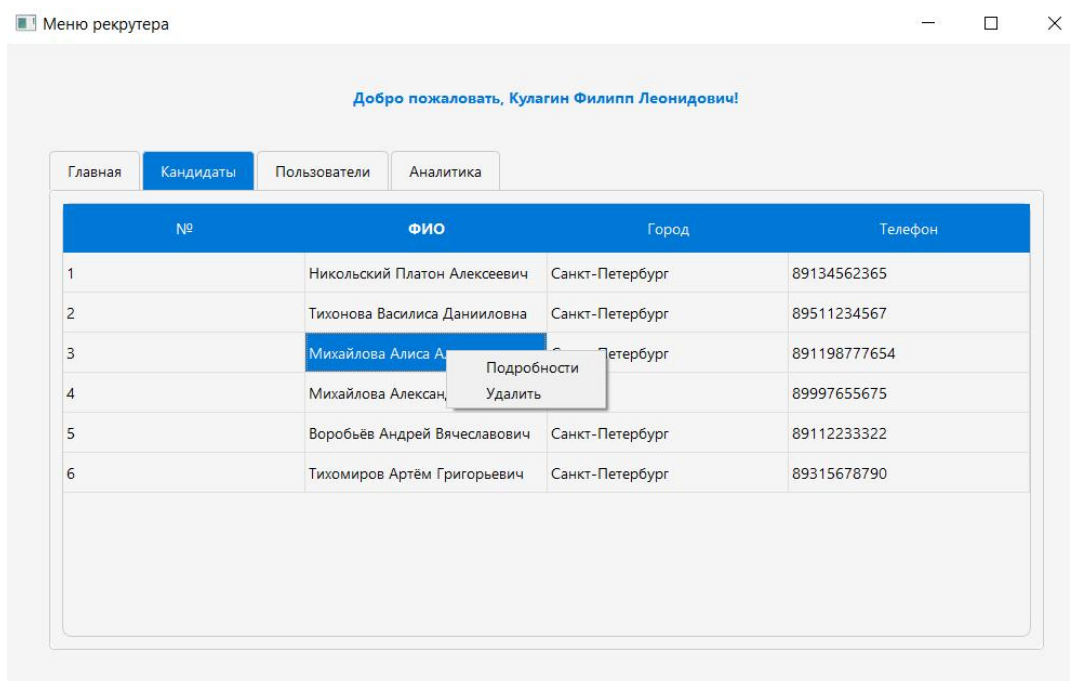


Рисунок 18 - Контекстное меню к таблице на вкладке «Кандидаты»

Кандидат Михайлова Алиса Александровна

Результаты кандидата: Михайлова Алиса Александровна

| | № | Статус | Баллы |
|---|--------|---------|-------|
| 1 | Тест 1 | пройден | 8/10 |
| 2 | Тест 2 | пройден | 5/10 |
| 3 | Тест 3 | пройден | 6/10 |
| 4 | Тест 4 | пройден | 4/10 |
| 5 | Тест 5 | пройден | 5/10 |

Рисунок 19 - Пример подробностей тестируемого

Выбирая в контекстном меню «Подробности», открывается окно с результатами всех пройденных тестов выбранного кандидата (рисунок 19). Если в контекстном меню выбрать «Удалить», будет предложено подтвердить удаление (рисунок 20).

Меню рекрутера

Добро пожаловать, Кулагин Филипп Леонидович!

Главная Кандидаты Пользователи Аналитика

| № | ФИО | Город | Телефон |
|---|------------------------------|-----------------|--------------|
| 1 | Н... | | 89134562365 |
| 2 | Т... | | 89511234567 |
| 3 | М... | | 891198777654 |
| 4 | М... | | 89997655675 |
| 5 | Ворожьев Андрей Вячеславович | Санкт-Петербург | 89112233322 |
| 6 | Тихомиров Артём Григорьевич | Санкт-Петербург | 89315678790 |

Подтверждение
Вы уверены, что хотите удалить этого кандидата?
Yes No

Рисунок 20 - Подтверждение удаления тестируемого

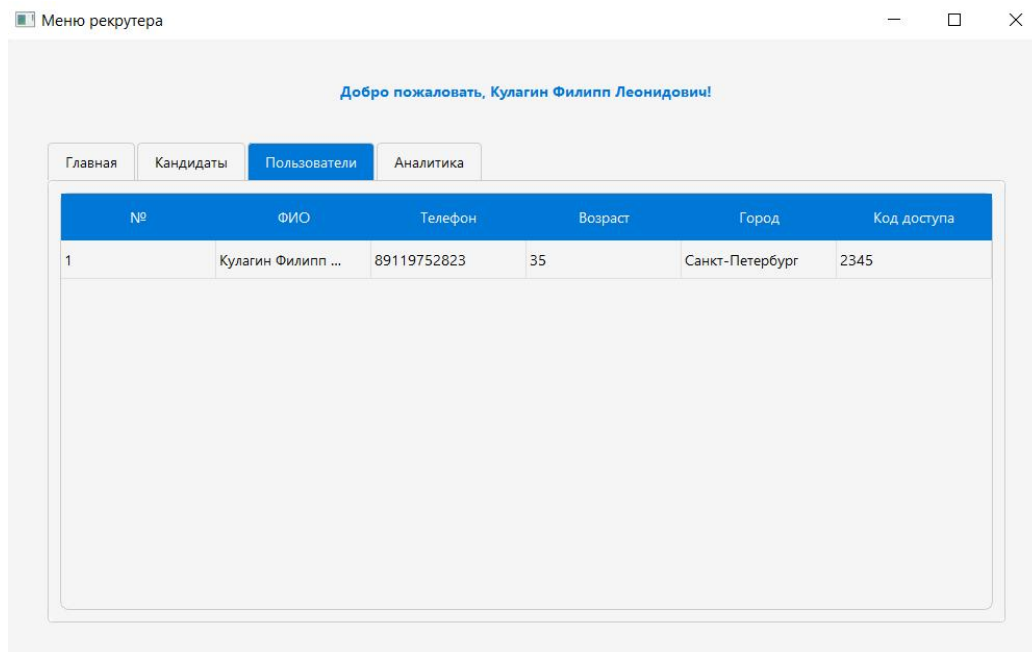


Рисунок 21 - Вкладка «Пользователи»

На рисунке 21 выведена таблица с данными обо всех зарегистрированных рекрутерах. В этой таблице также есть контекстное меню (рисунок 22), предлагающее две опции: «Добавить» и «Удалить»

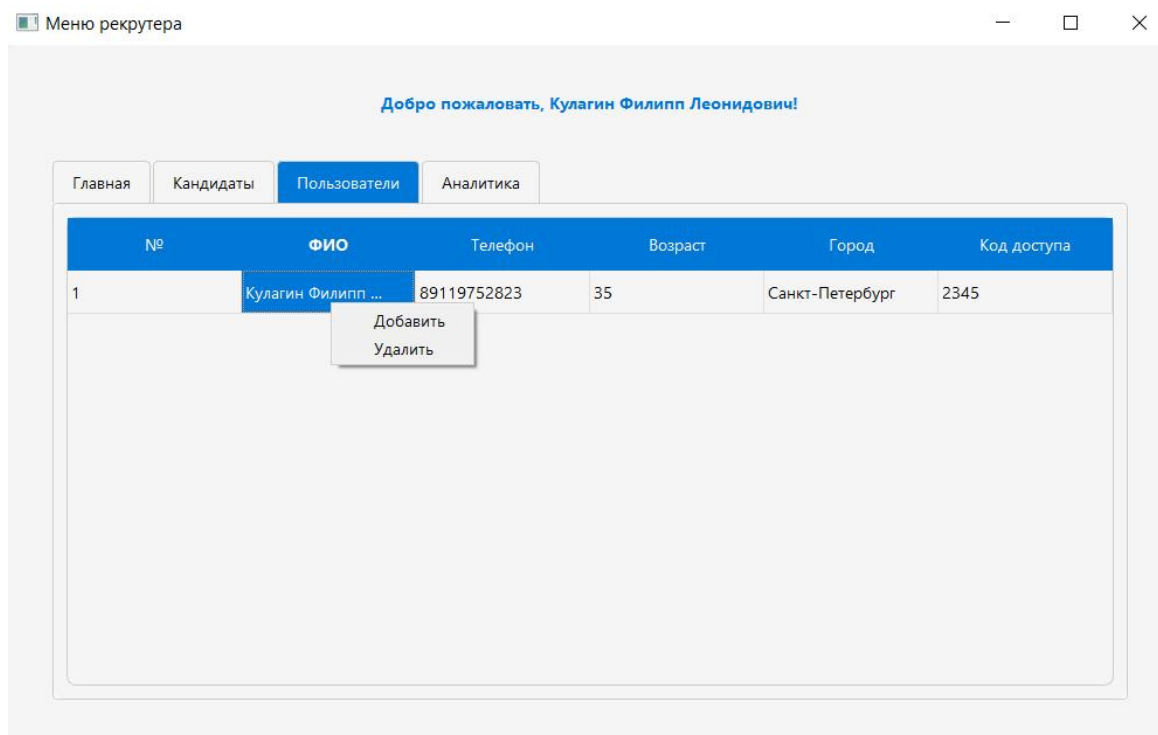


Рисунок 22 - Контекстное меню в таблице на вкладке «Пользователи»

Добавить рекрутера

Добавление нового сотрудника

Введите ФИО

Введите возраст

Введите город

Введите код доступа

Введите номер телефона

Добавить

Рисунок 23 - Окно добавления нового пользователя (рекрутера)

Как показано на рисунке 23, при выборе «Добавить» открывается окно-форма для заполнения данных. При правильном заполнении в БД будет создана запись о новом рекрутере.

При выборе удалить будет предложено подтвердить удаление (рисунок 24).

Меню рекрутера

Добро пожаловать, Кулагин Филипп Леонидович!

Главная Кандидаты Пользователи Аналитика

| № | ФИО | Телефон | Возраст | Город | Код доступа |
|---|---------|---------|---------|-------|-------------|
| 1 | Кулагин | | | | 2345 |

Подтверждение удаления

Вы уверены, что хотите удалить сотрудника Кулагин Филипп Леонидович?

Yes No

Рисунок 24 - Удаление рекрутера

Далее на вкладке «Аналитика» отображается подсчёт результатов тестирования кандидатов с предложением показать результат, а также возможность настроить таблицу приоритетов (рисунок 25).

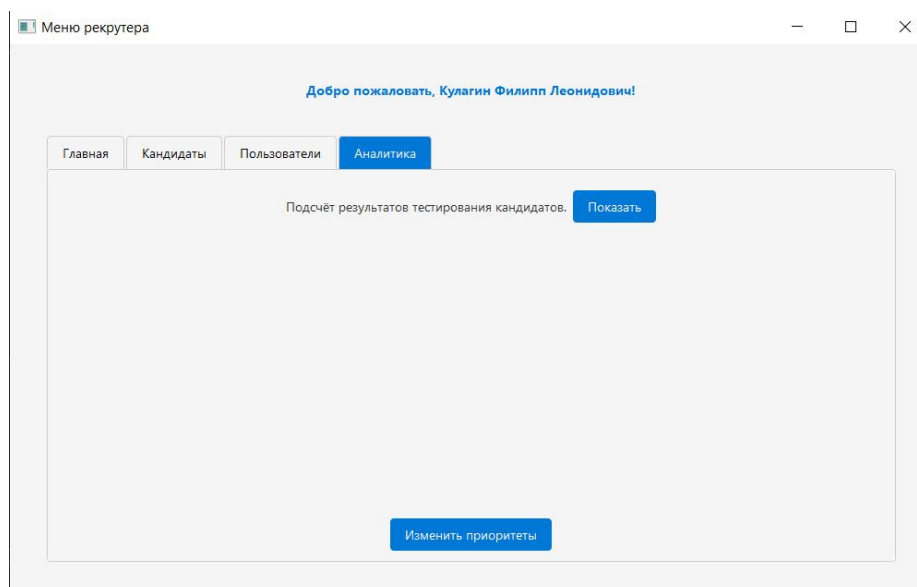


Рисунок 25 - Вкладка «Аналитика»

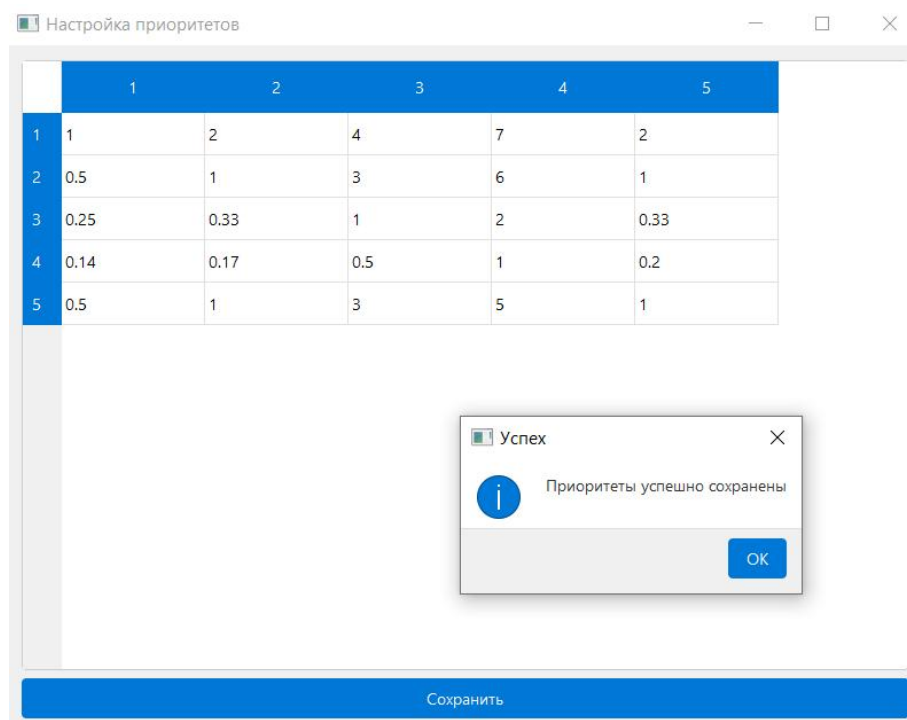


Рисунок 26 - Настройка приоритетов

При нажатии кнопки «Изменить приоритеты» открывается окно с таблицей для настройки (рисунок 26). Чтобы изменить значение ячейки, достаточно двойного нажатия на неё и записи нового значения. Чтобы сохранить изменения, необходимо нажать «Сохранить». Если введены корректные значения, придёт уведомление об успешности сохранения.

Аналитика

Аналитика результатов

| | № | ФИО | Результат |
|---|---|--------------------------------|-----------|
| 1 | 1 | Никольский Платон Алексеевич | 8.24 |
| 2 | 2 | Тихонова Василиса Данииловна | 2.43 |
| 3 | 3 | Михайлова Алиса Александровна | 6.23 |
| 4 | 4 | Михайлова Александра Фёдоровна | 5.43 |
| 5 | 5 | Воробьёв Андрей Вячеславович | 5.44 |
| 6 | 6 | Тихомиров Артём Григорьевич | 9.12 |

Рекомендуемый кандидат: Тихомиров Артём Григорьевич

Рисунок 27 - Отображение результата расчётов с использованием МАИ

Нажимая кнопку «Показать», открывается окно с табличным представлением результата расчётов с использованием МАИ (рисунок 27). Также для удобства в нижней части окна сразу пишется ФИО кандидата, получившего наибольший результат. В данном случае с тестовыми исходными данными, лучшим оказался кандидат №6 - Тихомиров Артём Григорьевич.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы был проведён анализ существующих математических моделей, применяемых для формализации процессов подбора и оценки персонала. В частности, рассмотрен метод нечётких деревьев для работы с нечёткими данными, метод анализа иерархий, оценивающий относительную важность критериев посредством парных сравнений, а также метод бинарного выбора для принятия решения в условиях двойственности исхода. Акцент сделан на возможностях их адаптации к специфике кадрового менеджмента. Анализ позволил определить ключевые факторы, влияющие на эффективность процессов подбора и оценки, а также выявить ограничения существующих методов.

В рамках практической части работы мною была достигнута поставленная цель: за счёт оптимизации оценки кандидатов и автоматизации процесса проведения тестирования была повышена эффективность подбора персонала. Для этого была разработана информационная система, состоящая из базы данных, которая хранит информацию о кандидатах, рекрутерах, результатов тестах и сами тесты; программы с интерфейсом взаимодействия с пользователями, включающей в себя проведение тестирования; модели, разработанной с применением метода анализа иерархий, для выявления лучшего кандидата. Для достижения поставленной цели были выполнены следующие задачи:

- провести анализ существующих подходов к подбору и оценке персонала в кадровом менеджменте и выявить проблемы;
- предложить методы, позволяющие повысить эффективность процесса подбора кандидата на должность;
- Разработать информационную систему оценки кандидатов на должность в IT-индустрии.

В заключение следует отметить, что разработанная информационная система является практически значимым инструментом для оптимизации процессов подбора и оценки персонала. Данная система может быть использована для повышения эффективности кадровых решений и снижения рисков, связанных с наймом неквалифицированных сотрудников.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванова О. Э. Управление человеческими ресурсами: концепция и методология оценки: учебно-методическое пособие / О. Э. Иванова, Д. Н. Корнеев, Н. Ю. Корнеева — Челябинск: Изд-во ЗАО «Библиотека А. Миллера», 2018. — 145 с.
2. Арнаут М. Н., Митрофанова Т. В. Кадровый менеджмент: сущность, подходы к трактовке, модели // Азимут научных исследований: экономика и управление. — 2018. Т. 7. №1 (22). — С.22—25.
3. Федеральная служба государственной статистики : официальный сайт. — Москва. — URL: https://rosstat.gov.ru/labour_force (дата обращения: 27.05.2025).
4. В В. Бут, Б М. Мусаева, А А. Перетяцько Проблема нехватки высококвалифицированных кадров в России // Деловой вестник предпринимателя. 2024. №2 (16). — С.15—17.
5. Натейкина Ю. О. Выявление проблем в процессе подбора персонала организации и пути их решения // European research. 2016. №1 (12). — С.79—81.
6. Логутова Т. Г., Ващенко В. В. Теоретико-практические аспекты процесса рекрутинга // The Scientific Heritage. 2017. №8-2 (8). — С.7—11.
7. Тихонов А. И. Современные методы оценки кандидатов при подборе персонала // Московский экономический журнал. 2020. №5. — С.631—637.
8. Каштанова Е. В. Современные методы адаптации персонала // УПИРР. 2019. №5. — С.34—40.
9. Кузнецов, Е. И. Анализ методов принятия решений для задач множественного выбора альтернатив / Е. И. Кузнецов // Математическое и

программное обеспечение вычислительных систем : межвузовский сборник научных трудов / Рязанский государственный радиотехнический университет. – Рязань : Индивидуальный предприниматель Коняхин Александр Викторович, 2020. – С. 45-47. – EDN VGSHQZ.

10. Воробьева М. В. Анализ методов многокритериального принятия решений // Региональная и отраслевая экономика. 2022. №1. — С.24—28.

11. Зелинская М. В., Пронин Е. С. Системный подход при отборе персонала: Основные этапы и критерии // Научный журнал КубГАУ. 2015. №108. — С.1093—1106.

12. Мельников О. Ю., Світлична М. В. Разработка системы поддержки принятия решений на основе метода нечетких деревьев для выбора претендента на вакантную должность в отделе промышленного предприятия // Вестник экономической науки Украины. 2016. №2 (31). — С.130—135.

13. Петриченко Г. С., Крицкая Л. М., Петриченко Д. Г. Методика оценки компетентности кандидатов при отборе на вакантную должность // Экономика. Информатика. 2019. №1. — С.130—137.

14. Зинченко Алексей Алексеевич Оценка персонала с использованием бинарной регрессии // Финансы: теория и практика. 2015. №2 (86). — С.135—141.

ПРИЛОЖЕНИЕ 1

| | ID_question [PK] integer | question text | answer1 text | answer2 text | answer3 text | answer4 text | correct_answer smallint |
|----|-----------------------------|----------------------|------------------------|-------------------------|---------------------|-------------------------------------|----------------------------|
| 1 | 1 | Выберите правиль... | Модуль os предна... | Модуль os исполь... | Модуль os пре... | Модуль os необходим для вы... | 3 |
| 2 | 2 | Выберите правиль... | datetime.datetime.t... | datetime.date(mont... | datetime.dateti... | datetime.timedelta(days="два") | 3 |
| 3 | 3 | Как модуль re исп... | Модуль re в Python... | Модуль re использ... | Модуль re авто... | Модуль re нужен для шифров... | 1 |
| 4 | 4 | Найдите вариант, ... | json.parse() | json.encode() | json.load() | json.jsonify() | 3 |
| 5 | 5 | Функциональность... | Counter - это функ... | Counter - это класс... | Counter в моду... | Counter реализует специализ... | 2 |
| 6 | 6 | Найдите пример и... | itertools.combine(A... | itertools.superzip([... | itertools.groupe... | itertools.pairwise([1, 2, 3, 4, 5]) | 4 |
| 7 | 7 | Как используется ... | Модуль sys предос... | Модуль sys испол... | Модуль sys ис... | Модуль sys предназначен дл... | 1 |
| 8 | 8 | Какова роль моду... | Модуль math пред... | Модуль math коде... | Модуль math и... | Модуль math позволяет рабо... | 1 |
| 9 | 9 | Найдите пример и... | random.order(my_li... | random.round(num) | random.randint... | random.array([1+1', '2*3', '10/2]) | 3 |
| 10 | 10 | Для чего предназн... | Для управления р... | Для работы с URL-... | Для создания ... | Для анализа HTML-страниц и... | 2 |

Рисунок 27 - Таблица «py_theory2» с тестовыми данными

| | ID_question [PK] integer | question text | answer1 text | answer2 text | answer3 text | answer4 text | correct_answer smallint |
|----|-----------------------------|----------------------|------------------------|--------------------|-----------------------|----------------------|----------------------------|
| 1 | 1 | Какой менеджер ... | conda | poetry | pip | art | 3 |
| 2 | 2 | Какой файл испол... | requirements.txt | environment.yml | Pipfile | pyproject.toml | 4 |
| 3 | 3 | Какой менеджер ... | pip | conda | pipenv | setup.py | 3 |
| 4 | 4 | В какой ситуации ... | Когда требуется упр... | Когда необходи... | Когда нужно устано... | Когда необходимо ... | 3 |
| 5 | 5 | Какое преимущес... | Позволяет указыва... | Содержит списо... | Используется для у... | Упрощает процесс... | 1 |
| 6 | 6 | Что означает "раз... | Установка всех дос... | Определение и у... | Удаление неисполь... | Проверка наличия... | 2 |
| 7 | 7 | Какой метод пере... | Прямое копировани... | Заморозка зави... | Экспорт списка пак... | Копирование всег... | 2 |
| 8 | 8 | Какое преимущес... | Упрощает создание ... | Гарантирует мак... | Автоматически упр... | Ускоряет процесс ... | 2 |
| 9 | 9 | Что может быть п... | Несовместимость в... | Несовместимос... | Жестко закодирова... | Отсутствие необхо... | 3 |
| 10 | 10 | Какой инструмент... | pip | venv | virtualenv | pipenv | 4 |

Рисунок 28 - Таблица «py_theory3» с тестовыми данными

| | ID_question [PK] integer | question text | answer1 text | answer2 text | answer3 text | answer4 text | correct_answer smallint |
|----|-----------------------------|------------------------|--------------------|-----------------|-----------------|-------------------|----------------------------|
| 1 | 1 | Какое ключевое сл... | def | async | thread | process | 2 |
| 2 | 2 | Что такое Event Loo... | Механизм, упра... | Основной по... | Механизм ... | Драйвер для р... | 1 |
| 3 | 3 | Какое ключевое сл... | sleep | wait | await | pause | 3 |
| 4 | 4 | В каких типах задач... | CPU-bound зада... | GPU-bound э... | Memory-bo... | I/O-bound зада... | 4 |
| 5 | 5 | Что представляет с... | Функция обратн... | Запланирова... | Объект, пре... | Класс для созд... | 2 |
| 6 | 6 | Что делает функция... | Запускает все з... | Создает нов... | Запускает ... | Отменяет все з... | 3 |
| 7 | 7 | Какой из приведен... | Более эффектив... | Более прост... | Подходит д... | Чистый и читае... | 3 |
| 8 | 8 | Какую функцию вы... | Создает новый ... | Запускает у... | Создает но... | Компилирует к... | 3 |
| 9 | 9 | Какое утверждение ... | Asyncio может н... | Использован... | Для работ... | Asyncio автома... | 3 |
| 10 | 10 | Для чего используе... | Для завершения... | Для отмены ... | Для принуд... | Для перезапус... | 2 |

Рисунок 29 - Таблица «py_theory4» с тестовыми данными

| | ID_question [PK] integer | question text | answer1 text | answer2 text | answer3 text | answer4 text | correct_answer smallint |
|----|-----------------------------|-------------------------|---------------------|-----------------|------------------|------------------------|----------------------------|
| 1 | 1 | Дана последователь... | 1000 | 1200 | 725 | 975 | 1 |
| 2 | 2 | Дана последователь... | 141 | 148 | 159 | 174 | 2 |
| 3 | 3 | Дана последователь... | 3/4 | 5/6 | 11/12 | 1 | 3 |
| 4 | 4 | Петя, Вася и Коля за... | ни у кого не сов... | Пети и Васи | Васи и Коли | Пети и Коли | 4 |
| 5 | 5 | На прямой отметили ... | 12 | 16 | 13 | 14 | 2 |
| 6 | 6 | Участников конфере... | Менее четверти ... | На третьем э... | Около 25% все... | Меньше 25 человек р... | 2 |
| 7 | 7 | Во время распродаж... | 5 | 6 | 7 | 8 | 4 |
| 8 | 8 | В школе 800 ученико... | 112 | 114 | 116 | 118 | 1 |
| 9 | 9 | 15 января планирует... | 4 266 000 | 4 166 000 | 4 155 000 | 3 995 000 | 1 |
| 10 | 10 | 15 января планирует... | 133 000 | 136 000 | 139 000 | 142 000 | 2 |

Рисунок 30 - Таблица «logic» с тестовыми данными

| | ID_candidate integer | ID_question integer | answer smallint |
|----|-------------------------|------------------------|--------------------|
| 1 | 1 | 1 | 3 |
| 2 | 1 | 2 | 3 |
| 3 | 1 | 3 | 1 |
| 4 | 1 | 4 | 3 |
| 5 | 1 | 5 | 2 |
| 6 | 1 | 6 | 4 |
| 7 | 1 | 7 | 1 |
| 8 | 1 | 8 | 2 |
| 9 | 1 | 9 | 2 |
| 10 | 1 | 10 | 1 |
| 11 | 2 | 1 | 3 |
| 12 | 2 | 2 | 3 |
| 13 | 2 | 3 | 1 |
| 14 | 2 | 4 | 2 |
| 15 | 2 | 5 | 1 |
| 16 | 2 | 6 | 3 |
| 17 | 2 | 7 | 4 |
| 18 | 2 | 8 | 4 |
| 19 | 2 | 9 | 4 |
| 20 | 2 | 10 | 4 |
| 21 | 3 | 1 | 3 |

Рисунок 31 - Таблица «candidates_answers_theory2» с тестовыми данными

| | ID_candidate integer | ID_question integer | answer smallint |
|----|-------------------------|------------------------|--------------------|
| 1 | 1 | 1 | 3 |
| 2 | 1 | 2 | 4 |
| 3 | 1 | 3 | 3 |
| 4 | 1 | 4 | 3 |
| 5 | 1 | 5 | 1 |
| 6 | 1 | 6 | 2 |
| 7 | 1 | 7 | 2 |
| 8 | 1 | 8 | 1 |
| 9 | 1 | 9 | 2 |
| 10 | 1 | 10 | 2 |
| 11 | 2 | 1 | 3 |
| 12 | 2 | 2 | 1 |
| 13 | 2 | 3 | 1 |
| 14 | 2 | 4 | 1 |
| 15 | 2 | 5 | 4 |
| 16 | 2 | 6 | 4 |
| 17 | 2 | 7 | 4 |
| 18 | 2 | 8 | 4 |
| 19 | 2 | 9 | 1 |
| 20 | 2 | 10 | 1 |
| 21 | 3 | 1 | 3 |

Рисунок 32 - Таблица «candidates_answers_theory3» с тестовыми данными

| | ID_candidate integer | ID_question integer | answer smallint |
|----|-------------------------|------------------------|--------------------|
| 1 | 1 | 1 | 2 |
| 2 | 1 | 2 | 1 |
| 3 | 1 | 3 | 3 |
| 4 | 1 | 4 | 4 |
| 5 | 1 | 5 | 2 |
| 6 | 1 | 6 | 3 |
| 7 | 1 | 7 | 3 |
| 8 | 1 | 8 | 3 |
| 9 | 1 | 9 | 4 |
| 10 | 1 | 10 | 4 |
| 11 | 2 | 1 | 2 |
| 12 | 2 | 2 | 1 |
| 13 | 2 | 3 | 3 |
| 14 | 2 | 4 | 1 |
| 15 | 2 | 5 | 1 |
| 16 | 2 | 6 | 1 |
| 17 | 2 | 7 | 1 |
| 18 | 2 | 8 | 1 |
| 19 | 2 | 9 | 1 |
| 20 | 2 | 10 | 1 |
| 21 | 3 | 1 | 2 |
| -- | - | - | - |

Рисунок 33 - Таблица «candidates_answers_theory4» с тестовыми данными

| | ID_candidate integer | ID_question integer | answer integer |
|----|-------------------------|------------------------|-------------------|
| 1 | 1 | 1 | 1 |
| 2 | 1 | 2 | 2 |
| 3 | 1 | 3 | 3 |
| 4 | 1 | 4 | 4 |
| 5 | 1 | 5 | 2 |
| 6 | 1 | 6 | 2 |
| 7 | 1 | 7 | 4 |
| 8 | 1 | 8 | 4 |
| 9 | 1 | 9 | 4 |
| 10 | 1 | 10 | 4 |
| 11 | 2 | 1 | 1 |
| 12 | 2 | 2 | 2 |
| 13 | 2 | 3 | 1 |
| 14 | 2 | 4 | 1 |
| 15 | 2 | 5 | 1 |
| 16 | 2 | 6 | 4 |
| 17 | 2 | 7 | 4 |
| 18 | 2 | 8 | 4 |
| 19 | 2 | 9 | 4 |
| 20 | 2 | 10 | 4 |
| 21 | 3 | 1 | 1 |
| 22 | 3 | 2 | 2 |

Рисунок 34 - Таблица «candidates_answers_theory4» с тестовыми данными

ПРИЛОЖЕНИЕ 2

Тест 1.

Вопрос 2: Разница между списками (list) и кортежами (tuples). Когда лучше использовать кортежи?

- Список (list) - изменяемая последовательность, кортеж (tuple) - неизменяемая. Кортежи предпочтительнее, когда данные должны оставаться константой.
- Список (list) - последовательность данных, которые могут быть разного типа, кортеж (tuple) - последовательность данных одного типа. Кортежи предпочтительнее, когда работаешь с данными одного типа.
- Список (list) - содержит в себе пары «ключ - значение», кортеж (tuple) - последовательность данных. Кортежи предпочтительнее, когда используются данные, не требующие использования идентификационных ключей.
- Список (list) - последовательность данных, кортеж (tuple) - содержит в себе пары «ключ - значение». Кортежи предпочтительнее, когда используются данные, требующие использования идентификационных ключей для удобства.

Вопрос 3: Что такое генератор (generator)? Как он работает?

- Генератор - специальная функция, возвращающая итератор. Он «выдаёт» значения по запросу при работе с большими данными. За счёт этого можно обрабатывать большие объёмы данных, не занимая много памяти.
- Генератор - устройство, преобразующее механическую энергию в электрическую. Работает на основе закона электромагнитной индукции Фарадея.

- Генератор - часть компилятора, отвечающая за преобразование исходного кода программы в машинный код, понятный процессору. Он оптимизирует код и генерирует наиболее эффективные инструкции.

- Генератор – это специальный тип переменной в языке программирования, который может хранить только случайно сгенерированные числа. Он используется для создания псевдослучайных последовательностей

Вопрос 4: Что такое декораторы (decorators)? Где указан корректный пример его использования?

- Декораторы – это специальные объекты в Python, используемые для динамического создания экземпляров классов. Они указываются внутри определения класса и определяют логику инициализации. Пример:

```
def my_decorator(n):  
  
    for i in range(n):  
  
        yield i  
  
for num in my_decorator(5):  
  
    print(num)
```

- Декораторы – это встроенные функции Python, предназначенные для работы со строками. Они позволяют форматировать строки, преобразовывать регистр символов и выполнять другие операции. Пример:

```
numbers = [1, 2, 3, 4, 5]  
  
squares = [x**2 for x in numbers]  
  
print(squares)
```

- Декораторы – это специальные модули в Python, расширяющие функциональность языка. Они импортируются как обычные модули и предоставляют доступ к новым функциям и классам. Пример:

```
multiply = lambda x, y: x * y
```

```
print(multiply(5, 6))
```

- Декораторы – это функции, позволяющие изменять поведение функций или классов, оборачивая их и не изменяя содержимое. Пример:

```
def my_decorator(func):
```

```
    def wrapper():
```

```
        print("""До вызова функции.""")
```

```
        func()
```

```
        print("""После вызова функции.""")
```

```
    return wrapper
```

```
@my_decorator
```

```
def say_hello():
```

```
    print("""Привет!""")
```

```
say_hello()" 4
```

Вопрос 5: Что такое множественное наследование и какие проблемы оно может вызвать?

- Множественное наследование – это процесс создания нескольких экземпляров одного класса. Оно может вызвать проблемы с управлением памятью, так как каждый экземпляр занимает отдельную область памяти, и их количество может быстро увеличиваться, приводя к переполнению.

- Множественное наследование – это механизм сокрытия данных внутри класса, обеспечивающий инкапсуляцию. Проблемы могут возникнуть при неправильной реализации приватных переменных, что может привести к нарушению целостности данных.

- Множественное наследование – это способ реализации полиморфизма, когда один и тот же метод может вести себя по-разному в разных классах. Основная проблема – это сложность реализации динамического связывания, когда выбор конкретного метода происходит во время выполнения программы, что снижает производительность.

Множественное наследование – это возможность класса наследовать свойства и методы от нескольких родительских классов. Это позволяет объединять функциональность из разных источников, создавая более сложные и специализированные классы. Одна из возможных проблем - возникновение конфликтов имён, когда разные родительские классы имеют методы или атрибуты с одинаковыми именами.

Вопрос 6: Объясните разницу между `==` и `is` в Python.

- `==` сравнивает типы данных объектов, возвращая `True`, если типы совпадают, а `is` сравнивает значения объектов, игнорируя типы. `==` преобразует типы данных перед сравнением, а `is` не выполняет преобразование.

- `==` сравнивает значения объектов, проверяя, эквивалентны ли они, в то время как `is` проверяет идентичность объектов, то есть, указывают ли они на одну и ту же область памяти.

- `==` сравнивает ID объектов в памяти, возвращая `True`, если объекты находятся в одном и том же месте, а `is` сравнивает значения объектов, выполняя глубокое сравнение всех атрибутов.

- `==` сравнивает указатели на объекты, возвращая `True`, если указатели совпадают, а `is` сравнивает длины объектов, если они являются строками или списками.

Вопрос 7: Что такое GIL в Python и как он влияет на многопоточное программирование?

- GIL – это встроенная функция в Python, которая управляет созданием новых процессов. Она позволяет эффективно распределять задачи между разными ядрами процессора, при этом значительно ускоряя выполнение параллельных вычислений.

- GIL – это глобальная блокировка ввода-вывода, которая предотвращает одновременный доступ к файлам и сетевым ресурсам из разных потоков. Для управления потоками и синхронизации данных используют примитивы синхронизации, которые защищают критические секции кода от одновременного доступа.

- GIL – это механизм в CPython, который позволяет только одному потоку выполнять байт-код Python в один момент времени. Даже на многоядерных процессорах программа, использующая потоки, не сможет в полной мере воспользоваться преимуществами параллелизма.

- GIL – это инструмент отладки многопоточных приложений в Python. Отслеживает состояние потоков и выявляет возможные ошибки синхронизации.

Вопрос 9: Как обрабатываются исключения (exceptions)?

- Используются блоки `try...except`. Код, который может вызвать исключение, помещается в блок `try`. Если в этом блоке возникает исключение, управление передается соответствующему блоку `except`.

- Исключения игнорируются по умолчанию. Разработчик должен использовать оператор `suppress` перед каждой строкой кода, которая может вызвать исключение, чтобы подавить его. Иначе программа просто завершится с ошибкой.

- Исключения в Python автоматически обрабатываются глобальным обработчиком ошибок, который перехватывает все исключения и записывает их в лог-файл. Разработчик не может вмешиваться в этот процесс и не может обрабатывать исключения самостоятельно.

- Для обработки исключений используются операторы `if...else`. Перед выполнением потенциально опасной операции нужно проверить, не может ли она вызвать исключение, и, если да, то выполнить альтернативный код в блоке `else`.

Вопрос 10: Разница между позиционными и именованными аргументами в функциях. Как использовать `*args` и `**kwargs`?

- Позиционные аргументы - это аргументы, которые обязательно должны быть указаны при объявлении функции, в то время как именованные аргументы являются необязательными. `*args` используется для передачи обязательных аргументов (списком или кортежом), а `**kwargs` - для передачи необязательных аргументов.

- Позиционные аргументы используются только для передачи числовых значений, а именованные аргументы - для передачи строк. `*args` используется для передачи числовых аргументов в виде кортежа, а `**kwargs` - для передачи строк в виде словаря.

- Позиционные аргументы передаются в функцию на основе их позиции в списке аргументов при вызове функции. Именованные аргументы передаются с указанием имени аргумента и его значения, порядок при этом не важен. `*args` используется для передачи произвольного количества

позиционных аргументов в функцию в виде кортежа, а `**kwargs` - для передачи произвольного количества именованных аргументов в виде словаря.

- Позиционные аргументы - это глобальные переменные, видимые во всей программе, а именованные аргументы - локальные переменные, видимые только внутри функции. `*args` используется для передачи статических глобальных переменных, а `**kwargs` - для передачи динамических глобальных переменных.

Тест 2.

Вопрос 2: Выберите правильный пример использования модуля `datetime`, который НЕ приведёт к ошибке

- `datetime.datetime.today().strftime("%Y-%m-%d") + 1`.
- `datetime.date(month=13, day=1, year=2023)`.
- `datetime.datetime.now()`.
- `datetime.timedelta(days="два")`.

Вопрос 3: Как модуль `re` используется для работы с регулярными выражениями?

- Модуль `re` в Python предоставляет набор функций для работы с регулярными выражениями. Он позволяет выполнять поиск, замену и разбиение строк на основе заданных шаблонов. Пример: `re.search()` - функция поиска первого соответствия.

- Модуль `re` используется для проверки синтаксической правильности HTML-кода. Функции, такие как `re.validate()`, позволяют убедиться, что HTML соответствует стандартам. Пример: `re.validate("<p>Hello</p>")` - проверка соответствия стандартам HTML.

- Модуль `re` автоматически преобразует все строки в юникод и сравнивает их на предмет совпадений. Для использования регулярных выражений нужно импортировать модуль и указывать строки, которые нужно проверить. Пример: `re.contrast("строка", "регулярка")` - сравнение юникода символов.

- Модуль `re` нужен для шифрования данных. Функция `re.encrypt()` берет строку и регулярное выражение в качестве ключа и шифрует строку. Пример: `re.encrypt("текст", "ключ")` - шифрование текста по ключу.

Вопрос 4: Найдите вариант, в котором указана существующая функция модуля `json`.

- `json.parse()`.
- `json.encode()`.
- `json.load()`.
- `json.jsonify()`.

Вопрос 5: Функциональность модуля `collections`. Что такое `Counter` и как его можно использовать?

- `Counter` - это функция в модуле `collections`, которая используется для создания глубоких копий объектов. Она позволяет избежать проблем, связанных с изменением исходного объекта при работе с его копией.

- `Counter` - это класс, являющийся подклассом словаря (`dict`), который предназначен для подсчета количества вхождений элементов в последовательности.

- `Counter` в модуле `collections` используется для сортировки элементов в коллекции в порядке убывания или возрастания. Он принимает коллекцию в качестве аргумента и возвращает отсортированную версию.

- Counter реализует специализированные типы данных-контейнеров, предоставляющие альтернативу встроенным контейнерам общего назначения.

Вопрос 6: Найдите пример использования существующей функции из модуля itertools.

- `itertools.combine('ABCD', 2).`
- `itertools.superzip([1, 2, 3], [4, 5, 6]).`
- `itertools.grouper('ABCDEFGH', 3, fillvalue='x').`
- `itertools.pairwise([1, 2, 3, 4, 5]).`

Вопрос 7: Как используется модуль sys?

- Модуль sys предоставляет доступ к некоторым переменным и функциям, тесно взаимодействующим с интерпретатором Python и его окружением.
- Модуль sys используется для управления системными вызовами операционной системы, такими как создание процессов, управление памятью и работа с файловой системой.
- Модуль sys используется для работы с сетевыми соединениями и протоколами, такими как TCP/IP и UDP.
- Модуль sys предназначен для работы с системными регистрами и аппаратным обеспечением компьютера.

Вопрос 9: Найдите пример использования существующей функции из модуля random.

- `random.order(my_list).`
- `random.round(num).`

- `random.randint(1, 6)`.
- `random.array(['1+1', '2*3', '10/2'])`.

Вопрос 10: Для чего предназначен модуль `irllib`?

- Для управления ресурсами системы, такими как память и процессорное время, при взаимодействии.
- Для работы с URL-адресами, отправки HTTP-запросов и получения данных из интернета.
- Для создания и управления локальными веб-серверами и маршрутизации входящих HTTP-запросов.
- Для анализа HTML-страниц и извлечения данных, требующих сложной обработки и парсинга структуры веб-страниц.

Тест 3.

Вопрос 2: Какой файл используется для указания зависимостей в проекте, управляемом Poetry?

- `requirements.txt`.
- `environment.yml`.
- `Pipfile`.
- `pyproject.toml`.

Вопрос 3: Какой менеджер пакетов автоматически создаёт и управляет виртуальными окружениями, упрощая процесс изоляции проектов?

- `pip`.
- `conda`.

- `pipenv`.
- `setup.py`.

Вопрос 4: В какой ситуации наиболее целесообразно использовать `pip`?

- Когда требуется управлять сложными зависимостями на разных языках программирования.
- Когда необходимо установить пакеты, требующие системных библиотек, которые `pip` не может установить автоматически.
- Когда нужно установить Python-пакет, доступный в PyPI, и не требуются сложные системные зависимости.
- Когда необходимо автоматически создавать и управлять виртуальными окружениями.

Вопрос 5: Какое преимущество предоставляет файл `Pipfile.lock`, используемый `Pipenv`?

- Позволяет указывать точные версии всех зависимостей, гарантируя воспроизводимость окружения.
- Содержит список всех доступных пакетов в PyPI.
- Используется для указания не-Python зависимостей.
- Упрощает процесс публикации пакетов на PyPI.

Вопрос 6: Что означает «разрешение зависимостей» в контексте управления пакетами?

- Установка всех доступных версий пакета в систему.
- Определение и установка всех пакетов, необходимых для работы определённого пакета, включая их собственные зависимости.

- Удаление неиспользуемых пакетов из системы.
- Проверка наличия обновлений для установленных пакетов.

Вопрос 7: Какой метод переноса виртуального окружения считается наиболее надёжным для обеспечения воспроизводимости проекта?

- Прямое копирование директории виртуального окружения (zip/tar).
- Заморозка зависимостей в requirements.txt и установка на новой машине.
- Экспорт списка пакетов через pip list и ручная установка.
- Копирование всего жесткого диска с проектом.

Вопрос 9: Что может быть проблемой при прямом копировании виртуального окружения между компьютерами?

- Несовместимость версий Python.
- Несовместимость архитектур процессоров.
- Жестко закодированные пути в окружении, специфичные для исходной системы.
- Отсутствие необходимых библиотек на новом компьютере.

Вопрос 10: Какой инструмент позволяет разделять зависимости, необходимые для разработки, от зависимостей, необходимых для запуска приложения?

- pip.
- venv.
- virtualenv.

- `pipenv`.

Тест 4.

Вопрос 2: Что такое Event Loop в контексте `asyncio`?

- Механизм, управляющий выполнением корутин и обработкой событий.
- Основной поток выполнения программы.
- Механизм для запуска нескольких потоков.
- Драйвер для работы с графическим интерфейсом.

Вопрос 3: Какое ключевое слово используется для приостановки выполнения корутины и ожидания результата свободного объекта?

- `sleep`.
- `wait`.
- `await`.
- `pause`.

Вопрос 4: В каких типах задач `asyncio` обеспечивает наибольший прирост производительности?

- CPU-bound задачах (вычислительно интенсивные задачи).
- GPU-bound задачах (задачи, использующие графический процессор).
- Memory-bound задачах (задачи, интенсивно использующие память).
- I/O-bound задачах (задачи, связанные с вводом-выводом, например, сетевые запросы).

Вопрос 5: Что представляет собой объект `Task` в `asyncio`?

- Функция обратного вызова, вызываемая после завершения операции.
- Запланированная корутина для выполнения.
- Объект, представляющий собой блокировку для синхронизации потоков.
- Класс для создания пользовательских исключений.

Вопрос 6: Что делает функция `asyncio.gather(*tasks)`?

- Запускает все задачи последовательно, одну за другой.
- Создаёт новые потоки для каждой задачи.
- Запускает все задачи параллельно и дожидается завершения всех задач.
- Отменяет все задачи.

Вопрос 7: Какой из приведённых вариантов НЕ является преимуществом `asyncio`?

- Более эффективное использование ресурсов для I/O-bound задач.
- Более простая реализация параллельных вычислений, чем с потоками.
- Подходит для всех типов задач, включая CPU-bound.
- Чистый и читаемый код с использованием `async` и `await`.

Вопрос 9: Какое утверждение является верным касательно использования `asyncio` с синхронными библиотеками?

- `Asyncio` может напрямую использовать любые синхронные библиотеки без каких-либо изменений.

- Использование синхронных библиотек в корутинах всегда приводит к полной блокировке event loop.

- Для работы с синхронными библиотеками внутри asyncio необходимо использовать asyncio.to_thread(), asyncio.run_in_executor() или аналогичные механизмы для запуска их в отдельном потоке или процессе, чтобы избежать блокировки event loop.

- Asyncio автоматически преобразует синхронные библиотеки в асинхронные.

Вопрос 10: Для чего используется метод task.cancel() у объекта Task в asyncio?

- Для завершения выполнения программы.
- Для отмены запланированной корутины, связанной с Task, если она ещё не завершена.
- Для принудительной остановки event loop.
- Для перезапуска корутины с начала.

Тест 5.

Вопрос 2: Дана последовательность, в которой пропущено три числа: 123, 129, _, _, _, 174, 186, 201, 204, 210, 213, 219. Какое из приведённых ниже чисел НЕ может быть частью последовательности?

- 141.
- 148.
- 159.
- 174.

Вопрос 3: Дана последовательность: $1/6, 1/4, 1/3, 5/12, 1/2, \dots$ Какое число будет стоять на десятом месте?

- $3/4$.
- $5/6$.
- $1/12$.
- 1.

Вопрос 4: Петя, Вася и Коля записали в ряд по 100 чисел. У Пети пятое число равно 12, и каждое число, начиная со второго, на два больше левого соседа. У Васи первое и третье число равны 4 и 6 соответственно, а каждое число, кроме крайних, вдвое меньше суммы его соседей. А Коля просто записывал периметры прямоугольников шириной в одну клетку: сначала – периметр прямоугольника длиной в одну клетку, потом – длиной в две клетки, и так далее (сторона каждой клетки равна 1). У кого из мальчиков совпали записанные ряды чисел?

- ни у кого не совпали.
- Пети и Васи.
- Васи и Коли.
- Пети и Коли.

Вопрос 5: На прямой отметили 100 точек так, что расстояние между любыми соседними точками равно 7. Какой номер будет иметь точка с координатой 110, если координата первой точки равна 5?

- 12.
- 16.
- 13.

- 14.

Вопрос 6: Участников конференции разместили в гостинице в одноместных номерах, расположенных на этажах со второго по пятый. Количество номеров на этажах представлено на круговой диаграмме. Какое утверждение относительно расселения верно, если в гостинице разместились 150 участников?

- Менее четверти всех участников разместились на втором этаже.
- На третьем этаже разместилось более чем в 2 раза больше участников, чем на втором.
- Около 25% всех участников конференции разместились на пятом этаже.
- Меньше 25 человек разместились на четвёртом этаже.

Вопрос 7: Во время распродажи Витя купил 6 одинаковых по цене футболок со скидкой 60%. Сколько таких футболок он мог бы купить на ту же сумму, если бы скидка составила 70%?

- 5.
- 6.
- 7.
- 8.

Вопрос 9: 15 января планируется взять кредит в банке на сумму 3,6 млн рублей на 36 месяцев. Условия его возврата таковы: 1-го числа каждого месяца долг возрастает на 1 % по сравнению с концом предыдущего месяца; со 2-го по 14-е число каждого месяца необходимо выплатить часть долга; 15-го числа каждого месяца долг должен быть на одну и ту же величину меньше долга на

15-е число предыдущего месяца. Какую сумму (в рублях) нужно вернуть банку за весь срок кредитования?

- 4 266 000.
- 4 166 000.
- 4 155 000.
- 3 995 000.

Вопрос 10: 15 января планируется взять кредит в банке на сумму 3,6 млн рублей на 36 месяцев. Условия его возврата таковы: 1-го числа каждого месяца долг возрастает на 1 % по сравнению с концом предыдущего месяца; со 2-го по 14-е число каждого месяца необходимо выплатить часть долга; 15-го числа каждого месяца долг должен быть на одну и ту же величину меньше долга на 15-е число предыдущего месяца. Сколько рублей составит первый платёж?

- 133 000.
- 136 000.
- 139 000.
- 142 000.

ПРИЛОЖЕНИЕ 3

Файл «main.py»:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton,
QVBoxLayout, QHBoxLayout, QMessageBox, QFrame
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont
from server import add_candidate, get_candidate_id_by_access_code,
get_recruiter_id_by_access_code
from menu import MenuWindow
from menu_recruiter import MenuRecruiterWindow

# Общие стили для приложения
STYLE_SHEET = """
QWidget {
    font-family: 'Segoe UI', Arial;
    font-size: 12px;
}
QLabel {
    color: #333333;
}
QLineEdit {
    padding: 8px;
    border: 1px solid #CCCCCC;
    border-radius: 4px;
    background-color: white;
    min-height: 20px;
}
QLineEdit:focus {
    border: 1px solid #0078D7;
}
QPushButton {
    padding: 8px 16px;
    background-color: #0078D7;
    color: white;
    border: none;
    border-radius: 4px;
    min-height: 20px;
}
QPushButton:hover {
    background-color: #106EBE;
}
QPushButton:pressed {
    background-color: #005A9E;
}
QFrame {
    background-color: white;
    border-radius: 8px;
}
"""
```



```

class WelcomeWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Добро пожаловать")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем заголовок "Добро пожаловать"
        self.title_label = QLabel("Добро пожаловать")
        font = QFont('Segoe UI', 24, QFont.Bold)
        self.title_label.setFont(font)
        self.title_label.setStyleSheet("color: #0078D7; margin: 20px;")

        # Создаем поле для ввода кода авторизации
        self.auth_code_label = QLabel("Введите код авторизации:")
        self.auth_code_label.setFont(QFont('Segoe UI', 12))
        self.auth_code_input = QLineEdit()
        self.auth_code_input.setPlaceholderText("Введите ваш код доступа")
        self.auth_code_input.setMinimumWidth(300)

        # Кнопка авторизации
        self.auth_button = QPushButton("Авторизоваться")
        self.auth_button.setFont(QFont('Segoe UI', 12))
        self.auth_button.setMinimumWidth(200)
        self.auth_button.clicked.connect(self.authorize)

        # Кнопка регистрации
        self.register_button = QPushButton("Регистрация")
        self.register_button.setFont(QFont('Segoe UI', 12))
        self.register_button.setFixedWidth(150)
        self.register_button.clicked.connect(self.open_registration)

        # Создаем основной вертикальный layout
        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.setContentsMargins(40, 40, 40, 40)

        # Создаем горизонтальный layout для заголовка и кнопки регистрации
        header_layout = QHBoxLayout()
        header_layout.addWidget(self.title_label, alignment=Qt.AlignCenter)
        header_layout.addWidget(self.register_button, alignment=Qt.AlignRight | Qt.AlignTop)
        layout.addLayout(header_layout)

```

```

# Добавляем отступ
layout.addSpacing(40)

# Создаем контейнер для формы авторизации
auth_frame = QFrame()
auth_frame.setStyleSheet("""
    QFrame {
        background-color: white;
        border-radius: 8px;
        padding: 20px;
    }
""")
auth_layout = QVBoxLayout(auth_frame)
auth_layout.setSpacing(15)

# Добавляем поля в контейнер
auth_layout.addWidget(self.auth_code_label)
auth_layout.addWidget(self.auth_code_input)
auth_layout.addWidget(self.auth_button, alignment=Qt.AlignCenter)

layout.addWidget(auth_frame)
layout.addStretch()

main_frame.setLayout(layout)

# Создаем основной layout для окна
main_layout = QVBoxLayout(self)
main_layout.setContentsMargins(0, 0, 0, 0)
main_layout.addWidget(main_frame)

def authorize(self):
    auth_code = self.auth_code_input.text()
    if not auth_code:
        QMessageBox.warning(self, "Ошибка", "Пожалуйста, введите код авторизации")
        return

    # Проверяем код доступа кандидата
    success_candidate, result_candidate = get_candidate_id_by_access_code(auth_code)
    if success_candidate:
        QMessageBox.information(self, "Успех", "Авторизация успешна!")
        self.menu_window = MenuWindow(result_candidate)
        self.menu_window.show()
        self.close()
        return

    # Проверяем код доступа рекрутера
    success_recruiter, result_recruiter = get_recruiter_id_by_access_code(auth_code)
    if success_recruiter:
        QMessageBox.information(self, "Успех", "Авторизация успешна!")
        self.menu_recruiter_window = MenuRecruiterWindow(result_recruiter)
        self.menu_recruiter_window.show()

```

```

        self.close()
        return

    QMessageBox.warning(self, "Ошибка", "Неверный код авторизации")

def open_registration(self):
    self.registration_window = RegistrationWindow()
    self.registration_window.show()

class RegistrationWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Регистрация нового пользователя")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем заголовок "Регистрация"
        self.title_label = QLabel("Регистрация")
        font = QFont('Segoe UI', 24, QFont.Bold)
        self.title_label.setFont(font)
        self.title_label.setStyleSheet("color: #0078D7; margin: 20px;")

        # Создаем контейнер для формы
        form_frame = QFrame()
        form_frame.setStyleSheet("""
            QFrame {
                background-color: white;
                border-radius: 8px;
                padding: 20px;
            }
        """)

        # Создаем поля ввода
        self.create_input_field("ФИО:", "full_name")
        self.create_input_field("Возраст:", "age")
        self.create_input_field("Город:", "city")
        self.create_input_field("Телефон:", "phone")
        self.create_input_field("Код доступа:", "access_code")

        # Кнопка регистрации
        self.register_button = QPushButton("Зарегистрироваться")
        self.register_button.setFont(QFont('Segoe UI', 12))
        self.register_button.setMinimumWidth(200)

```

```

self.register_button.clicked.connect(self.register)

# Создаем основной layout
layout = QVBoxLayout()
layout.setSpacing(20)
layout.setContentsMargins(40, 40, 40, 40)

# Добавляем заголовок
layout.addWidget(self.title_label, alignment=Qt.AlignCenter)

# Добавляем отступ
layout.addSpacing(20)

# Создаем layout для формы
form_layout = QVBoxLayout(form_frame)
form_layout.setSpacing(15)

# Добавляем все поля ввода
for label, input_field in self.input_fields.items():
    field_layout = QHBoxLayout()
    field_layout.addWidget(label)
    field_layout.addWidget(input_field)
    form_layout.addLayout(field_layout)

# Добавляем кнопку регистрации
form_layout.addWidget(self.register_button, alignment=Qt.AlignCenter)

layout.addWidget(form_frame)
main_frame.setLayout(layout)

# Создаем основной layout для окна
main_layout = QVBoxLayout(self)
main_layout.setContentsMargins(0, 0, 0, 0)
main_layout.addWidget(main_frame)

def create_input_field(self, label_text, field_name, is_password=False):
    if not hasattr(self, 'input_fields'):
        self.input_fields = {}

    label = QLabel(label_text)
    label.setFont(QFont('Segoe UI', 12))
    input_field = QLineEdit()
    input_field.setMinimumWidth(300)
    if is_password:
        input_field.setEchoMode(QLineEdit.Password)

    self.input_fields[label] = input_field
    setattr(self, f'{field_name}_input', input_field)

def show_registration_result(self, success, result):
    if success:
        QMessageBox.information(self, "Успех", f"Регистрация успешно завершена! Ваш код

```

```

доступа: {result}")
    self.close()
else:
    QMessageBox.warning(self, "Ошибка", f"Ошибка при регистрации: {result}")

def register(self):
    full_name = self.full_name_input.text()
    age = self.age_input.text()
    city = self.city_input.text()
    phone = self.phone_input.text()
    access_code = self.access_code_input.text()

    if not all([full_name, age, city, phone, access_code]):
        QMessageBox.warning(self, "Ошибка", "Пожалуйста, заполните все поля")
        return
    success, result = add_candidate(full_name, age, city, access_code, phone)
    self.show_registration_result(success, result)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = WelcomeWindow()
    window.show()
    sys.exit(app.exec_())

```

Файл «menu.py».

```

import sys
from PyQt5.QtWidgets import (QApplication, QWidget, QLabel, QPushButton, QVBoxLayout,
                             QTabWidget,
                             QButtonGroup, QMessageBox, QRadioButton, QTableWidgetItem, QHeaderView,
                             QHBoxLayout, QTableWidgetItem, QFrame)
from PyQt5.QtGui import QPixmap, QFont
from PyQt5.QtCore import Qt
from server import (get_candidate_fio_by_id, get_question_by_id_theory1, save_answer,
                    get_question_by_id_theory2, save_answer_theory2,
                    get_question_by_id_theory3, save_answer_theory3,
                    get_question_by_id_theory4, save_answer_theory4,
                    get_answers_list_theory1, get_answers_list_theory2,
                    get_answers_list_theory3, get_answers_list_theory4,
                    get_question_text_by_id_theory1, get_question_text_by_id_theory2,
                    get_question_text_by_id_theory3, get_question_text_by_id_theory4,
                    get_answer_by_id_theory1, get_answer_by_id_theory2,
                    get_answer_by_id_theory3, get_answer_by_id_theory4,
                    check_answers_exist_theory1, check_answers_exist_theory2,
                    check_answers_exist_theory3, check_answers_exist_theory4,
                    check_answers_exist_logic, get_question_by_id_logic,
                    save_answer_logic, get_answers_list_logic,
                    get_question_text_by_id_logic, get_answer_by_id_logic)

```

Общие стили для приложения

```

STYLE_SHEET = ""
QWidget {
    font-family: 'Segoe UI', Arial;
    font-size: 14px;
}
QLabel {
    color: #333333;
}
QPushButton {
    padding: 10px 20px;
    background-color: #0078D7;
    color: white;
    border: none;
    border-radius: 4px;
    min-height: 30px;
    font-size: 14px;
}
QPushButton:hover {
    background-color: #106EBE;
}
QPushButton:pressed {
    background-color: #005A9E;
}
QRadioButton {
    padding: 10px;
    spacing: 10px;
    font-size: 14px;
}
QRadioButton::indicator {
    width: 20px;
    height: 20px;
}
QTableWidget {
    border: 1px solid #CCCCCC;
    border-radius: 4px;
    background-color: white;
    font-size: 14px;
}
QTableWidget::item {
    padding: 10px;
}
QHeaderView::section {
    background-color: #F0F0F0;
    padding: 10px;
    border: 1px solid #CCCCCC;
    font-weight: bold;
    font-size: 14px;
}
QFrame {
    background-color: white;
    border-radius: 8px;
}

```

"""

```
class TestWindow(QWidget):
    def __init__(self, ID_candidate):
        super().__init__()
        self.setWindowTitle("Тест 1")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # ID кандидата
        self.ID_candidate = ID_candidate
        self.test_number = 1

        # Проверяем, проходил ли кандидат тест
        success, message = check_answers_exist_theory1(ID_candidate)
        if success:
            QMessageBox.warning(self, "Предупреждение", "Предупреждение: Вы уже проходили
этот тест. Повторное прохождение невозможно.")
            self.close()
            return

        # Массив для хранения ответов пользователя
        self.user_answers = []

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем основной layout
        self.main_layout = QVBoxLayout()
        self.main_layout.setSpacing(20)
        self.main_layout.setContentsMargins(40, 40, 40, 40)
        main_frame.setLayout(self.main_layout)

        # Создаем основной layout для окна
        window_layout = QVBoxLayout(self)
        window_layout.setContentsMargins(0, 0, 0, 0)
        window_layout.addWidget(main_frame)

        # Показываем начальный экран
        self.show_start_screen()

    def show_start_screen(self):
        # Очищаем текущий layout
        self.clear_layout()

        # Создаем контейнер для содержимого
```

```

content_frame = QFrame()
content_frame.setStyleSheet("""
    QFrame {
        background-color: white;
        border-radius: 8px;
        padding: 20px;
    }
""")
content_layout = QVBoxLayout(content_frame)
content_layout.setSpacing(20)

title = QLabel("Данный тест состоит из десяти вопросов, к каждому прилагается четыре
варианта ответа. Этим тестом мы хотим проверить ваши знания основ языка Python.
Нажмите кнопку \"Начать\", когда будете готовы.")
title.setFont(QFont('Segoe UI', 14))
title.setAlignment(Qt.AlignCenter)
title.setWordWrap(True)
title.setStyleSheet("color: #0078D7;")

start_button = QPushButton("Начать")
start_button.setFont(QFont('Segoe UI', 12))
start_button.setMinimumWidth(200)
start_button.clicked.connect(self.start_test)

content_layout.addWidget(title)
content_layout.addWidget(start_button, alignment=Qt.AlignCenter)

self.main_layout.addWidget(content_frame)

def clear_layout(self):
    # Удаляем все виджеты из layout
    while self.main_layout.count():
        child = self.main_layout.takeAt(0)
        if child.widget():
            child.widget().deleteLater()

def start_test(self):
    # Запускаем тест
    self.current_question = 1
    self.show_question(self.current_question)

def show_question(self, question_number):
    success, question_data = get_question_by_id_theory1(question_number)

    if success:
        # Очищаем текущий layout
        self.clear_layout()

        # Создаем контейнер для вопроса
        question_frame = QFrame()
        question_frame.setStyleSheet("""
            QFrame {

```



```

        background-color: white;
        border-radius: 8px;
        padding: 20px;
    }
    """
question_layout = QVBoxLayout(question_frame)
question_layout.setSpacing(20)

# Создаем и настраиваем label для вопроса
question_label = QLabel(question_data['question'])
question_label.setFont(QFont('Segoe UI', 14))
question_label.setWordWrap(True)
question_label.setAlignment(Qt.AlignCenter)
question_label.setStyleSheet("color: #0078D7;")

# Создаем radio buttons для вариантов ответов
self.answer_group = QButtonGroup()
answers_frame = QFrame()
answers_layout = QVBoxLayout(answers_frame)
answers_layout.setSpacing(10)

for i in range(1, 5):
    radio_button = QRadioButton(question_data[f'answer{i}'])
    radio_button.setFont(QFont('Segoe UI', 12))
    radio_button.setProperty('answer_value', i)
    self.answer_group.addButton(radio_button)
    answers_layout.addWidget(radio_button)

# Кнопка для следующего вопроса
next_button = QPushButton("Следующий вопрос")
next_button.setFont(QFont('Segoe UI', 12))
next_button.setMinimumWidth(200)
next_button.clicked.connect(self.next_question)

question_layout.addWidget(question_label)
question_layout.addWidget(answers_frame)
question_layout.addWidget(next_button, alignment=Qt.AlignCenter)

self.main_layout.addWidget(question_frame)

def next_question(self):
    # Проверяем, выбран ли вариант ответа
    selected_button = self.answer_group.checkedButton()
    if not selected_button:
        QMessageBox.warning(self, "Предупреждение", "Необходимо выбрать вариант ответа.")
        return

    # Сохраняем выбранный ответ вместе с ID кандидата и ID вопроса
    answer_value = selected_button.property('answer_value')
    self.user_answers.append((self.ID_candidate, self.current_question, answer_value))

    self.current_question += 1

```

```

if self.current_question <= 10:
    self.show_question(self.current_question)
else:
    self.save_and_finish_test()

def save_and_finish_test(self):
    # Сохраняем все ответы в базу данных
    for i in range(len(self.user_answers)):
        id_candidate, id_question, answer = self.user_answers[i]
        success, message = save_answer(id_candidate, id_question, answer)
        if not success:
            QMessageBox.warning(self, "Ошибка", f"Ошибка при сохранении ответа {i+1}:"
{message}")
        return

    # Очищаем текущий layout
    self.clear_layout()

    # Создаем контейнер для сообщения
    message_frame = QFrame()
    message_frame.setStyleSheet("""
        QFrame {
            background-color: white;
            border-radius: 8px;
            padding: 20px;
        }
    """)
    message_layout = QVBoxLayout(message_frame)

    # Показываем сообщение о завершении
    message = QLabel("Поздравляем с прохождением теста!\nВаши ответы записаны.")
    message.setFont(QFont('Segoe UI', 14))
    message.setWordWrap(True)
    message.setAlignment(Qt.AlignCenter)
    message.setStyleSheet("color: #0078D7;")

    message_layout.addWidget(message)
    self.main_layout.addWidget(message_frame)

```

(Для остальных тестов реализованы аналогичные классы.)

```

class ResultsWindow1(QWidget):
    def __init__(self, ID_candidate):
        super().__init__()
        self.setWindowTitle("Результаты теста 1")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # ID кандидата
        self.ID_candidate = ID_candidate

        # Получаем ответы кандидата

```

```

success, answers = get_answers_list_theory1(ID_candidate)

# Создаем основной контейнер
main_frame = QFrame()
main_frame.setStyleSheet("""
    QFrame {
        background-color: #F5F5F5;
        border-radius: 8px;
    }
""")

# Создаем основной layout
main_layout = QVBoxLayout()
main_layout.setSpacing(20)
main_layout.setContentsMargins(40, 40, 40, 40)
main_frame.setLayout(main_layout)

# Создаем основной layout для окна
window_layout = QVBoxLayout(self)
window_layout.setContentsMargins(0, 0, 0, 0)
window_layout.addWidget(main_frame)

# Заголовок
title = QLabel("Результаты теста 1")
title.setFont(QFont('Segoe UI', 20, QFont.Bold))
title.setAlignment(Qt.AlignCenter)
title.setStyleSheet("color: #0078D7; margin-bottom: 20px;")
main_layout.addWidget(title)

# Создаем таблицу
table = QTableWidget()
table.setColumnCount(2)
table.setRowCount(10)
table.setHorizontalHeaderLabels(["Вопрос", "Ответ"])
table.horizontalHeader().setSectionResizeMode(0, QHeaderView.Stretch)
table.horizontalHeader().setSectionResizeMode(1, QHeaderView.Stretch)
table.setStyleSheet("""
    QTableWidget {
        background-color: white;
        border-radius: 8px;
        gridline-color: #E0E0E0;
    }
    QHeaderView::section {
        background-color: #0078D7;
        color: white;
        padding: 8px;
        border: none;
        font-size: 14px;
    }
    QTableWidget::item {
        padding: 8px;
        font-size: 14px;
    }
""")

```

```
}  
""")
```

```
# Заполняем таблицу  
for i, (question_id, answer_id) in enumerate(answers):  
    # Получаем текст вопроса  
    success, question_text = get_question_text_by_id_theory1(question_id)  
    if success:  
        question_item = QTableWidgetItem(question_text)  
        question_item.setFlags(question_item.flags() & ~Qt.ItemIsEditable)  
        table.setItem(i, 0, question_item)  
  
    # Получаем текст ответа  
    success, answer_text = get_answer_by_id_theory1(question_id, answer_id)  
    if success:  
        answer_item = QTableWidgetItem(answer_text)  
        answer_item.setFlags(answer_item.flags() & ~Qt.ItemIsEditable)  
        table.setItem(i, 1, answer_item)  
  
# Устанавливаем высоту строк  
for i in range(table.rowCount()):  
    table.setRowHeight(i, 60)  
  
main_layout.addWidget(table)  
  
# Кнопка закрытия  
close_button = QPushButton("Закрыть")  
close_button.setFont(QFont('Segoe UI', 14))  
close_button.setMinimumWidth(250)  
close_button.setMinimumHeight(50)  
close_button.clicked.connect(self.close)  
main_layout.addWidget(close_button, alignment=Qt.AlignCenter)
```

(Для вывода результатов остальных тестов созданы аналогичные классы.)

```
class MenuWindow(QWidget):  
    def __init__(self, ID_candidate):  
        super().__init__()  
        self.setWindowTitle("Меню")  
        self.setGeometry(100, 100, 800, 600)  
        self.setStyleSheet(STYLE_SHEET)  
  
        # ID кандидата  
        self.ID_candidate = ID_candidate  
  
        # Получаем ФИО кандидата  
        success, fio = get_candidate_fio_by_id(ID_candidate)  
        if not success:  
            QMessageBox.warning(self, "Ошибка", f"Ошибка при получении данных кандидата:  
{fio}")  
        return
```

```

# Создаем основной контейнер
main_frame = QFrame()
main_frame.setStyleSheet("""
    QFrame {
        background-color: #F5F5F5;
        border-radius: 8px;
    }
""")

# Создаем основной layout
layout = QVBoxLayout()
layout.setSpacing(20)
layout.setContentsMargins(40, 40, 40, 40)

# Создаем заголовок
title = QLabel(f'Добро пожаловать, {fio}!')
title.setFont(QFont('Segoe UI', 28, QFont.Bold))
title.setAlignment(Qt.AlignCenter)
title.setStyleSheet("color: #0078D7; margin: 20px;")

# Создаем вкладки
self.tabs = QTabWidget()
self.tabs.setStyleSheet("""
    QTabWidget::pane {
        border: 1px solid #CCCCCC;
        border-radius: 8px;
        background-color: white;
    }
    QTabBar::tab {
        background-color: #F0F0F0;
        border: 1px solid #CCCCCC;
        border-bottom: none;
        border-top-left-radius: 4px;
        border-top-right-radius: 4px;
        padding: 8px 16px;
        margin-right: 2px;
    }
    QTabBar::tab:selected {
        background-color: white;
        border-bottom: 1px solid white;
    }
    QTabBar::tab:hover {
        background-color: #E0E0E0;
    }
""")

# Создаем первую вкладку (Главная)
self.tab1 = QWidget()
tab1_layout = QVBoxLayout()
tab1_layout.addWidget(title)
self.tab1.setLayout(tab1_layout)

```

```

# Создаем вторую вкладку (Тесты)
self.tab2 = QWidget()
tab2_layout = QVBoxLayout()
tab2_layout.setSpacing(15)

# Создаем контейнер для кнопок
buttons_frame = QFrame()
buttons_frame.setStyleSheet("""
    QFrame {
        background-color: white;
        border-radius: 8px;
        padding: 20px;
    }
""")
buttons_layout = QVBoxLayout(buttons_frame)
buttons_layout.setSpacing(15)

# Создаем кнопки
test1_button = QPushButton("Тест 1")
test2_button = QPushButton("Тест 2")
test3_button = QPushButton("Тест 3")
test4_button = QPushButton("Тест 4")
test5_button = QPushButton("Тест 5")

# Настраиваем кнопки
for button in [test1_button, test2_button, test3_button, test4_button, test5_button]:
    button.setFont(QFont('Segoe UI', 14))
    button.setMinimumWidth(250)
    button.setMinimumHeight(50)

# Подключаем обработчики
test1_button.clicked.connect(self.open_test1)
test2_button.clicked.connect(self.open_test2)
test3_button.clicked.connect(self.open_test3)
test4_button.clicked.connect(self.open_test4)
test5_button.clicked.connect(self.open_test5)

# Добавляем кнопки в layout
buttons_layout.addWidget(test1_button, alignment=Qt.AlignCenter)
buttons_layout.addWidget(test2_button, alignment=Qt.AlignCenter)
buttons_layout.addWidget(test3_button, alignment=Qt.AlignCenter)
buttons_layout.addWidget(test4_button, alignment=Qt.AlignCenter)
buttons_layout.addWidget(test5_button, alignment=Qt.AlignCenter)

tab2_layout.addWidget(buttons_frame)
self.tab2.setLayout(tab2_layout)

# Создаем третью вкладку (Ответы)
self.tab3 = QWidget()
tab3_layout = QVBoxLayout()

```

```

# Создаем контейнер для таблицы
table_frame = QFrame()
table_frame.setStyleSheet("""
    QFrame {
        background-color: white;
        border-radius: 8px;
        padding: 20px;
    }
""")
table_layout = QVBoxLayout(table_frame)

# Создаем таблицу
table = QTableWidgetItem()
table.setRowCount(5)
table.setColumnCount(2)
table.setHorizontalHeaderLabels(["Доступные тесты", "Ответы"])
table.setStyleSheet("""
    QTableWidgetItem {
        border: none;
        background-color: white;
        font-size: 14px;
    }
    QTableWidgetItem::item {
        padding: 10px;
    }
    QHeaderView::section {
        background-color: #F0F0F0;
        padding: 10px;
        border: 1px solid #CCCCCC;
        font-weight: bold;
        font-size: 14px;
    }
""")

# Устанавливаем минимальную высоту строк
table.verticalHeader().setDefaultSectionSize(80)

# Растягиваем столбцы по содержимому
header = table.horizontalHeader()
header.setSectionResizeMode(0, QHeaderView.Stretch)
header.setSectionResizeMode(1, QHeaderView.Stretch)

# Заполняем таблицу
for i in range(5):
    # Добавляем название теста
    test_label = QLabel(f"Тест {i+1}")
    test_label.setAlignment(Qt.AlignCenter)
    test_label.setFont(QFont('Segoe UI', 16, QFont.Bold))
    test_label.setStyleSheet("color: #0078D7;")
    table.setCellWidget(i, 0, test_label)

# Добавляем кнопку "посмотреть"

```

```

view_button = QPushButton("Посмотреть")
view_button.setFont(QFont('Segoe UI', 14))
view_button.setMinimumWidth(180)
view_button.setMinimumHeight(45)

# Подключаем соответствующий обработчик для каждой кнопки
if i == 0:
    view_button.clicked.connect(self.show_answers_theory1)
elif i == 1:
    view_button.clicked.connect(self.show_answers_theory2)
elif i == 2:
    view_button.clicked.connect(self.show_answers_theory3)
elif i == 3:
    view_button.clicked.connect(self.show_answers_theory4)
else:
    view_button.clicked.connect(self.show_answers_theory5)

cell_widget = QWidget()
cell_layout = QHBoxLayout(cell_widget)
cell_layout.addWidget(view_button)
cell_layout.setAlignment(Qt.AlignCenter)
cell_layout.setContentsMargins(0, 0, 0, 0)
table.setCellWidget(i, 1, cell_widget)

# Настраиваем размеры столбцов и строк
table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
table.verticalHeader().setVisible(False)

table_layout.addWidget(table)
tab3_layout.addWidget(table_frame)
self.tab3.setLayout(tab3_layout)

# Добавляем вкладки
self.tabs.addTab(self.tab1, "Главная")
self.tabs.addTab(self.tab2, "Тесты")
self.tabs.addTab(self.tab3, "Ответы")

# Добавляем все в основной layout
layout.addWidget(self.tabs)
main_frame.setLayout(layout)

# Создаем основной layout для окна
window_layout = QVBoxLayout(self)
window_layout.setContentsMargins(0, 0, 0, 0)
window_layout.addWidget(main_frame)

def open_test1(self):
    self.test_window = TestWindow(self.ID_candidate)
    self.test_window.show()

def open_test2(self):
    self.test_window = TestWindow2(self.ID_candidate)

```



```

self.test_window.show()

def open_test3(self):
    self.test_window = TestWindow3(self.ID_candidate)
    self.test_window.show()

def open_test4(self):
    self.test_window = TestWindow4(self.ID_candidate)
    self.test_window.show()

def open_test5(self):
    self.test_window = TestWindow5(self.ID_candidate)
    self.test_window.show()

def show_answers_theory1(self):
    self.results_window = ResultsWindow1(self.ID_candidate)
    self.results_window.show()

def show_answers_theory2(self):
    self.results_window = ResultsWindow2(self.ID_candidate)
    self.results_window.show()

def show_answers_theory3(self):
    self.results_window = ResultsWindow3(self.ID_candidate)
    self.results_window.show()

def show_answers_theory4(self):
    self.results_window = ResultsWindow4(self.ID_candidate)
    self.results_window.show()

def show_answers_theory5(self):
    self.results_window = ResultsWindow5(self.ID_candidate)
    self.results_window.show()

```

Файл «menu_recruiter.py».

```

import sys
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout, QTabWidget, QLabel,
                             QTableWidgetItem,
                             QHeaderView, QPushButton, QHBoxLayout, QLineEdit, QMessageBox, QMenu,
                             QFrame)
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QFont

from server import (get_recruiter_fio_by_id, get_candidate_info_by_id, get_candidates_count,
                    delete_candidate_by_id,
                    check_answers_exist_theory1, check_answers_exist_theory2,
                    check_answers_exist_theory3,
                    check_answers_exist_theory4, check_answers_exist_logic, get_candidate_fio_by_id,
                    get_answers_list_by_candidate_theory1, get_answers_list_by_candidate_theory2,

```

```

        get_answers_list_by_candidate_theory3, get_answers_list_by_candidate_theory4,
        get_answers_list_by_candidate_logic, get_recruiters_info, add_recruiter,
        delete_recruiter_by_id, get_answers_listt_theory1, get_answers_listt_theory2,
        get_answers_listt_theory3, get_answers_listt_theory4, get_answers_listt_logic,
reorder_candidate_ids)
from analis import get_candidate_score
from priorities import get_priorities, set_priorities

# Глобальный стиль для всего приложения
STYLE_SHEET = """
QWidget {
    font-family: 'Segoe UI';
    font-size: 14px;
}
QPushButton {
    background-color: #0078D7;
    color: white;
    border: none;
    padding: 8px 16px;
    border-radius: 4px;
    font-size: 14px;
}
QPushButton:hover {
    background-color: #106EBE;
}
QPushButton:pressed {
    background-color: #005A9E;
}
QLineEdit {
    padding: 8px;
    border: 1px solid #CCCCCC;
    border-radius: 4px;
    background-color: white;
}
QLineEdit:focus {
    border: 1px solid #0078D7;
}
QTableWidget {
    border: 1px solid #CCCCCC;
    border-radius: 4px;
    background-color: white;
    gridline-color: #E0E0E0;
}
QHeaderView::section {
    background-color: #0078D7;
    color: white;
    padding: 8px;
    border: none;
    font-size: 14px;
}
QTableWidget::item {
    padding: 8px;

```

```

    }
    QLabel {
        color: #333333;
    }
}
"""

```

```

class DetailsWindow(QWidget):
    def __init__(self, id_candidate):
        super().__init__()
        success, fio = get_candidate_fio_by_id(id_candidate)
        self.setWindowTitle(f"Кандидат {fio}")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # ID кандидата
        self.id_candidate = id_candidate

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем основной layout
        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.setContentsMargins(40, 40, 40, 40)
        main_frame.setLayout(layout)

        # Создаем основной layout для окна
        window_layout = QVBoxLayout(self)
        window_layout.setContentsMargins(0, 0, 0, 0)
        window_layout.addWidget(main_frame)

        # Заголовок
        title = QLabel(f"Результаты кандидата: {fio}")
        title.setFont(QFont("Segoe UI", 20, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        title.setStyleSheet("color: #0078D7; margin-bottom: 20px;")
        layout.addWidget(title)

        # Создаем таблицу
        self.tests_table = QTableWidget()
        self.tests_table.setRowCount(5)
        self.tests_table.setColumnCount(3)
        self.tests_table.setHorizontalHeaderLabels(["№", "Статус", "Баллы"])
        self.tests_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

        # Устанавливаем высоту строк

```

```

for i in range(self.tests_table.rowCount()):
    self.tests_table.setRowHeight(i, 60)

# Заполняем первый столбец
test_names = ["Тест 1", "Тест 2", "Тест 3", "Тест 4", "Тест 5"] # Добавлен "Тест 5"
for i in range(5): # Изменено с 4 на 5
    self.tests_table.setItem(i, 0, QTableWidgetItem(test_names[i]))

# Заполняем второй и третий столбцы
check_functions = [
    check_answers_exist_theory1,
    check_answers_exist_theory2,
    check_answers_exist_theory3,
    check_answers_exist_theory4,
    check_answers_exist_logic
]

get_answers_functions = [
    get_answers_list_by_candidate_theory1,
    get_answers_list_by_candidate_theory2,
    get_answers_list_by_candidate_theory3,
    get_answers_list_by_candidate_theory4,
    get_answers_list_by_candidate_logic
]

for i in range(5): # Изменено с 4 на 5
    success, _ = check_functions[i](self.id_candidate)
    status = "пройден" if success else "не пройден"
    self.tests_table.setItem(i, 1, QTableWidgetItem(status))

    if success:
        # Получаем ответы кандидата
        success, answers = get_answers_functions[i](self.id_candidate)
        correct_count = 0

        if success:
            # Получаем правильные ответы
            success, correct_answers = get_answers_listt_theory1() if i == 0 else \
                get_answers_listt_theory2() if i == 1 else \
                get_answers_listt_theory3() if i == 2 else \
                get_answers_listt_theory4() if i == 3 else \
                get_answers_listt_logic()

            if success:
                # Сравниваем ответы кандидата с правильными ответами
                for user_answer, correct_answer in zip(answers, correct_answers):
                    if user_answer == correct_answer:
                        correct_count += 1

            self.tests_table.setItem(i, 2, QTableWidgetItem(f"{correct_count}/10"))
        else:
            self.tests_table.setItem(i, 2, QTableWidgetItem("Ошибка"))

```

```

        else:
            self.tests_table.setItem(i, 2, QTableWidgetItem("Ошибка"))
        else:
            self.tests_table.setItem(i, 2, QTableWidgetItem("---"))

    layout.addWidget(self.tests_table)
    self.setLayout(layout)

class AddRecruiterWindow(QWidget):
    dataChanged = pyqtSignal()

    def __init__(self, parent=None):
        super().__init__(parent=parent)
        self.parent_window = parent
        self.setWindowTitle("Добавить рекрутера")
        self.setGeometry(100, 100, 500, 400)
        self.setWindowModality(Qt.ApplicationModal)
        self.setWindowFlags(Qt.Window)
        self.setStyleSheet(STYLE_SHEET)

    # Создаем основной контейнер
    main_frame = QFrame()
    main_frame.setStyleSheet("""
        QFrame {
            background-color: #F5F5F5;
            border-radius: 8px;
        }
    """)

    # Создаем основной layout
    layout = QVBoxLayout()
    layout.setSpacing(20)
    layout.setContentsMargins(40, 40, 40, 40)
    main_frame.setLayout(layout)

    # Создаем основной layout для окна
    window_layout = QVBoxLayout(self)
    window_layout.setContentsMargins(0, 0, 0, 0)
    window_layout.addWidget(main_frame)

    # Заголовок
    title = QLabel("Добавление нового сотрудника")
    title.setFont(QFont('Segoe UI', 20, QFont.Bold))
    title.setAlignment(Qt.AlignCenter)
    title.setStyleSheet("color: #0078D7; margin-bottom: 20px;")
    layout.addWidget(title)

    # Создаем поля ввода
    self.fio_input = QLineEdit()
    self.fio_input.setPlaceholderText("Введите ФИО")
    self.fio_input.setMinimumHeight(40)

```

```

self.age_input = QLineEdit()
self.age_input.setPlaceholderText("Введите возраст")
self.age_input.setMinimumHeight(40)

self.city_input = QLineEdit()
self.city_input.setPlaceholderText("Введите город")
self.city_input.setMinimumHeight(40)

self.access_code_input = QLineEdit()
self.access_code_input.setPlaceholderText("Введите код доступа")
self.access_code_input.setMinimumHeight(40)

self.phone_input = QLineEdit()
self.phone_input.setPlaceholderText("Введите номер телефона")
self.phone_input.setMinimumHeight(40)

# Добавляем поля в layout
layout.addWidget(self.fio_input)
layout.addWidget(self.age_input)
layout.addWidget(self.city_input)
layout.addWidget(self.access_code_input)
layout.addWidget(self.phone_input)

# Кнопка добавления
self.add_button = QPushButton("Добавить")
self.add_button.setMinimumWidth(250)
self.add_button.setMinimumHeight(50)
self.add_button.clicked.connect(self.add_recruiter_clicked)
layout.addWidget(self.add_button, alignment=Qt.AlignCenter)

self.setLayout(layout)

# Центрируем окно относительно родительского окна
if parent:
    self.move(parent.x() + (parent.width() - self.width()) // 2,
              parent.y() + (parent.height() - self.height()) // 2)

def add_recruiter_clicked(self):
    fio = self.fio_input.text().strip()
    age = self.age_input.text().strip()
    city = self.city_input.text().strip()
    access_code = self.access_code_input.text().strip()
    phone_number = self.phone_input.text().strip()

    # Проверка на пустые поля
    if not all([fio, age, city, access_code, phone_number]):
        QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены!")
        return

    # Проверка возраста
    try:
        age = int(age)

```

```

    if age < 18 or age > 100:
        QMessageBox.warning(self, "Ошибка", "Возраст должен быть от 18 до 100 лет!")
        return
    except ValueError:
        QMessageBox.warning(self, "Ошибка", "Возраст должен быть числом!")
        return

    # Проверка телефона
    if not phone_number.replace('+', '').replace('-', '').replace(' ', '').isdigit():
        QMessageBox.warning(self, "Ошибка", "Номер телефона должен содержать только
цифры!")
        return

    success, message = add_recruiter(fio, age, city, access_code, phone_number)

    if success:
        QMessageBox.information(self, "Успех", "Рекрутер успешно добавлен!")
        self.dataChanged.emit() # Отправляем сигнал об изменении данных
        self.close() # Закрываем окно
    else:
        QMessageBox.warning(self, "Ошибка", f"Ошибка при добавлении рекрутера:
{message}")

class AnalyticsWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Аналитика")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем основной layout
        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.setContentsMargins(40, 40, 40, 40)
        main_frame.setLayout(layout)

        # Создаем основной layout для окна
        window_layout = QVBoxLayout(self)
        window_layout.setContentsMargins(0, 0, 0, 0)
        window_layout.addWidget(main_frame)

        # Заголовок
        title = QLabel("Аналитика результатов")

```

```

title.setFont(QFont('Segoe UI', 20, QFont.Bold))
title.setAlignment(Qt.AlignCenter)
title.setStyleSheet("color: #0078D7; margin-bottom: 20px;")
layout.addWidget(title)

# Получаем количество кандидатов
success, candidates_count = get_candidates_count()
print(f'Количество кандидатов: {candidates_count}')

# Создаем таблицу
self.results_table = QTableWidget()
self.results_table.setRowCount(candidates_count)
self.results_table.setColumnCount(3)
self.results_table.setHorizontalHeaderLabels(["№", "ФИО", "Результат"])
self.results_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

# Устанавливаем высоту строк
for i in range(self.results_table.rowCount()):
    self.results_table.setRowHeight(i, 60)

# Получаем результаты для каждого кандидата
results = []
max_result = 0
max_index = 0

for i in range(candidates_count):
    print(f'\nОбработка кандидата {i + 1}:')
    success, score = get_candidate_score(i + 1)
    print(f'Успех: {success}, Score: {score}')
    if success:
        results.append(score)
        if score > max_result:
            max_result = score
            max_index = i
    else:
        print(f'Ошибка при получении score: {score}')
        results.append(0)

print(f'\nМаксимальный результат: {max_result} у кандидата {max_index + 1}')

# Получаем ФИО рекомендуемого кандидата
success, recommended_fio = get_candidate_fio_by_id(max_index + 1)
print(f'ФИО рекомендуемого кандидата: {recommended_fio}')

# Заполняем таблицу
for i in range(candidates_count):
    # Номер кандидата
    self.results_table.setItem(i, 0, QTableWidgetItem(str(i + 1)))

    # ФИО кандидата
    success, fio = get_candidate_fio_by_id(i + 1)
    if success:

```



```

        self.results_table.setItem(i, 1, QTableWidgetItem(fio))
        print(f'Кандидат {i + 1}: {fio}, результат: {results[i]:.2f}')

    # Результат
    result_item = QTableWidgetItem(f'{results[i]:.2f}')
    result_item.setTextAlignment(Qt.AlignCenter)
    self.results_table.setItem(i, 2, result_item)

layout.addWidget(self.results_table)

# Добавляем информацию о рекомендуемом кандидате
if success and recommended_fio:
    recommended_label = QLabel(f'Рекомендуемый кандидат: {recommended_fio}')
    recommended_label.setFont(QFont('Segoe UI', 14, QFont.Bold))
    recommended_label.setStyleSheet("color: #0078D7; margin-top: 20px;")
    recommended_label.setAlignment(Qt.AlignCenter)
    layout.addWidget(recommended_label)

class PrioritySettingsWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Настройка приоритетов")
        self.setGeometry(100, 100, 800, 600)
        self.setStyleSheet(STYLE_SHEET)

    # Создаем таблицу приоритетов
    self.priorities_table = QTableWidgetItem()
    self.priorities_table.setRowCount(5)
    self.priorities_table.setColumnCount(5)

    # Заполняем таблицу текущими значениями
    current_priorities = get_priorities()
    for i in range(5):
        for j in range(5):
            item = QTableWidgetItem(str(current_priorities[i][j]))
            self.priorities_table.setItem(i, j, item)

    # Добавляем таблицу на форму
    layout = QVBoxLayout()
    layout.addWidget(self.priorities_table)

    # Добавляем кнопку сохранения
    save_button = QPushButton("Сохранить")
    save_button.clicked.connect(self.save_priorities)
    layout.addWidget(save_button)

    self.setLayout(layout)

    def save_priorities(self):
        try:
            new_priorities = []
            for i in range(5):

```

```

        row = []
        for j in range(5):
            value = float(self.priorities_table.item(i, j).text())
            row.append(value)
        new_priorities.append(row)
        set_priorities(new_priorities)
        QMessageBox.information(self, "Успех", "Приоритеты успешно сохранены")
    except Exception as e:
        QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить приоритеты: {str(e)}")

class MenuRecruiterWindow(QWidget):
    def __init__(self, ID_recruiter):
        super().__init__()
        self.ID_recruiter = ID_recruiter
        self.setWindowTitle("Меню рекрутера")
        self.setGeometry(100, 100, 1000, 600)
        self.setStyleSheet(STYLE_SHEET)

        # Создаем основной контейнер
        main_frame = QFrame()
        main_frame.setStyleSheet("""
            QFrame {
                background-color: #F5F5F5;
                border-radius: 8px;
            }
        """)

        # Создаем основной layout
        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.setContentsMargins(40, 40, 40, 40)
        main_frame.setLayout(layout)

        # Создаем основной layout для окна
        window_layout = QVBoxLayout(self)
        window_layout.setContentsMargins(0, 0, 0, 0)
        window_layout.addWidget(main_frame)

        # Получаем ФИО рекрутера
        success, fio = get_recruiter_fio_by_id(ID_recruiter)

        # Заголовок
        title = QLabel(f"Добро пожаловать, {fio}!")
        title.setFont(QFont('Segoe UI', 20, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        title.setStyleSheet("color: #0078D7; margin-bottom: 20px;")
        layout.addWidget(title)

        # Создаем вкладки
        self.tabs = QTabWidget()
        self.tabs.setStyleSheet("""
            QTabWidget::pane {

```

```

        border: 1px solid #CCCCCC;
        border-radius: 4px;
        background-color: white;
    }
    QTabBar::tab {
        background-color: #F5F5F5;
        border: 1px solid #CCCCCC;
        border-bottom: none;
        border-top-left-radius: 4px;
        border-top-right-radius: 4px;
        padding: 8px 16px;
        margin-right: 2px;
    }
    QTabBar::tab:selected {
        background-color: #0078D7;
        color: white;
    }
    QTabBar::tab:hover:!selected {
        background-color: #E0E0E0;
    }
}
"""
)

```

```

# Создаем layout для первой вкладки
tab1_layout = QVBoxLayout()

```

```

# Добавляем приветствие
self.welcome_label = QLabel(f"Здравствуйте, {fio}!")
font = self.welcome_label.font()
font.setPointSize(16)
font.setBold(True)
self.welcome_label.setFont(font)
self.welcome_label.setAlignment(Qt.AlignCenter)

```

```

tab1_layout.addWidget(self.welcome_label)
self.tab1 = QWidget()
self.tab1.setLayout(tab1_layout)

```

```

# Создаем layout для второй вкладки
tab2_layout = QVBoxLayout()

```

```

# Получаем количество кандидатов
success, candidates_count = get_candidates_count()

```

```

# Создаем таблицу
self.candidates_table = QTableWidgetItem()
self.candidates_table.setRowCount(candidates_count)
self.candidates_table.setColumnCount(4)
self.candidates_table.setHorizontalHeaderLabels(["№", "ФИО", "Город", "Телефон"])

```

```

# Заполняем таблицу
for i in range(candidates_count):
    # Номер кандидата

```

```

self.candidates_table.setItem(i, 0, QTableWidgetItem(str(i + 1)))

# Получаем информацию о кандидате
success, info = get_candidate_info_by_id(i + 1)
if success:
    self.candidates_table.setItem(i, 1, QTableWidgetItem(info['fio']))
    self.candidates_table.setItem(i, 2, QTableWidgetItem(info['city']))
    self.candidates_table.setItem(i, 3, QTableWidgetItem(info['number']))

# Настраиваем размеры столбцов
self.candidates_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
self.candidates_table.verticalHeader().setVisible(False)

# Включаем контекстное меню
self.candidates_table.setContextMenuPolicy(Qt.CustomContextMenu)
self.candidates_table.customContextMenuRequested.connect(self.show_context_menu)

tab2_layout.addWidget(self.candidates_table)

self.tab2 = QWidget()
self.tab2.setLayout(tab2_layout)

# Создаем layout для третьей вкладки
tab3_layout = QVBoxLayout()

# Создаем таблицу рекрутеров
self.recruiters_table = QTableWidgetItem()
self.recruiters_table.setColumnCount(6)
self.recruiters_table.setHorizontalHeaderLabels(["№", "ФИО", "Телефон", "Возраст",
"Город", "Код доступа"])

# Настраиваем размеры столбцов
self.recruiters_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
self.recruiters_table.verticalHeader().setVisible(False)

# Включаем контекстное меню для таблицы рекрутеров
self.recruiters_table.setContextMenuPolicy(Qt.CustomContextMenu)
self.recruiters_table.customContextMenuRequested.connect(self.show_recruiters_context_me
nu)

tab3_layout.addWidget(self.recruiters_table)
self.tab3 = QWidget()
self.tab3.setLayout(tab3_layout)

# Создаем layout для четвертой вкладки
tab4_layout = QVBoxLayout()

# Создаем контейнер для центрирования содержимого
center_container = QWidget()
center_layout = QVBoxLayout(center_container)
center_layout.setAlignment(Qt.AlignCenter)

```

```

# Создаем верхний layout для лейбла и кнопки "Показать"
top_layout = QHBoxLayout()

# Создаем лейбл с надписью
self.analytics_label = QLabel("Подсчёт результатов тестирования кандидатов.")
top_layout.addWidget(self.analytics_label)

# Создаем кнопку "Показать" и подключаем к ней обработчик
self.show_button = QPushButton("Показать")
self.show_button.clicked.connect(self.show_analytics)
top_layout.addWidget(self.show_button)

# Добавляем верхний layout в центрирующий контейнер
center_layout.addLayout(top_layout)

# Добавляем центрирующий контейнер в основной layout вкладки
tab4_layout.addWidget(center_container)

# Добавляем растягивающийся элемент для прижатия кнопки к низу
tab4_layout.addStretch()

# Добавляем кнопку "Изменить приоритеты" внизу
self.change_priorities_button = QPushButton("Изменить приоритеты")
self.change_priorities_button.setStyleSheet("""
    QPushButton {
        background-color: #0078D7;
        color: white;
        border: none;
        padding: 8px 16px;
        border-radius: 4px;
        margin-top: 20px;
    }
    QPushButton:hover {
        background-color: #005A9E;
    }
    """)
self.change_priorities_button.clicked.connect(self.show_priority_settings)
tab4_layout.addWidget(self.change_priorities_button, alignment=Qt.AlignCenter)

self.tab4 = QWidget()
self.tab4.setLayout(tab4_layout)

# Добавляем вкладки
self.tabs.addTab(self.tab1, "Главная")
self.tabs.addTab(self.tab2, "Кандидаты")
self.tabs.addTab(self.tab3, "Пользователи")
self.tabs.addTab(self.tab4, "Аналитика")

layout.addWidget(self.tabs)

# Загружаем данные рекрутеров при инициализации
self.load_recruiters()

```

```

def show_analytics(self):
    self.analytics_window = AnalyticsWindow()
    self.analytics_window.show()

def show_priority_settings(self):
    self.priority_settings_window = PrioritySettingsWindow()
    self.priority_settings_window.show()

def show_context_menu(self, pos):
    # Получаем индекс строки под курсором
    row = self.candidates_table.rowAt(pos.y())
    if row >= 0: # Если курсор находится над строкой
        menu = QMenu(self)
        details_action = menu.addAction("Подробности")
        delete_action = menu.addAction("Удалить")
        action = menu.exec_(self.candidates_table.viewport().mapToGlobal(pos))

        if action == details_action:
            # Получаем ID кандидата из первой колонки
            candidate_id = int(self.candidates_table.item(row, 0).text())
            # Открываем окно с подробностями
            self.details_window = DetailsWindow(candidate_id)
            self.details_window.show()

        elif action == delete_action:
            # Получаем ID кандидата из первой колонки
            candidate_id = int(self.candidates_table.item(row, 0).text())
            # Здесь можно добавить подтверждение удаления
            reply = QMessageBox.question(self, 'Подтверждение',
                                         'Вы уверены, что хотите удалить этого кандидата?',
                                         QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
            if reply == QMessageBox.Yes:
                # Удаляем кандидата из базы данных
                success, message = delete_candidate_by_id(candidate_id)
                if success:
                    # Удаляем строку из таблицы
                    self.candidates_table.removeRow(row)
                    QMessageBox.information(self, "Успех", message)
                else:
                    QMessageBox.warning(self, "Ошибка", message)

def show_recruiters_context_menu(self, pos):
    menu = QMenu(self)
    add_action = menu.addAction("Добавить")
    delete_action = menu.addAction("Удалить")

    action = menu.exec_(self.recruiters_table.viewport().mapToGlobal(pos))

    row = self.recruiters_table.rowAt(pos.y())

    if action == add_action:

```

```

self.add_recruiter_window = AddRecruiterWindow(self)
self.add_recruiter_window.dataChanged.connect(self.load_recruiters)
self.add_recruiter_window.show()
elif action == delete_action and row >= 0:
    # Получаем ID рекрутера из первой колонки
    recruiter_id = int(self.recruiters_table.item(row, 0).text())

    # Получаем ФИО рекрутера для отображения в сообщении
    recruiter_fio = self.recruiters_table.item(row, 1).text()

    # Запрашиваем подтверждение удаления
    reply = QMessageBox.question(
        self,
        'Подтверждение удаления',
        f'Вы уверены, что хотите удалить сотрудника {recruiter_fio}?',
        QMessageBox.Yes | QMessageBox.No,
        QMessageBox.No
    )

    if reply == QMessageBox.Yes:
        # Удаляем рекрутера из базы данных
        success, message = delete_recruiter_by_id(recruiter_id)
        success, message = reorder_candidate_ids()

        if success:
            # Обновляем таблицу
            self.load_recruiters()
            QMessageBox.information(self, "Успех", "Сотрудник успешно удален")
        else:
            QMessageBox.warning(self, "Ошибка", f"Не удалось удалить сотрудника:
{message}")

def load_recruiters(self):
    success, recruiters_info = get_recruiters_info()
    if success:
        # Очищаем таблицу перед загрузкой новых данных
        self.recruiters_table.clearContents()
        self.recruiters_table.setRowCount(0)

        # Устанавливаем количество строк
        self.recruiters_table.setRowCount(len(recruiters_info))

        # Заполняем таблицу данными
        for i, recruiter in enumerate(recruiters_info):
            self.recruiters_table.setItem(i, 0, QTableWidgetItem(str(i + 1)))
            self.recruiters_table.setItem(i, 1, QTableWidgetItem(str(recruiter['fio'])))
            self.recruiters_table.setItem(i, 2, QTableWidgetItem(str(recruiter['phone_number'])))
            self.recruiters_table.setItem(i, 3, QTableWidgetItem(str(recruiter['age'])))
            self.recruiters_table.setItem(i, 4, QTableWidgetItem(str(recruiter['city'])))
            self.recruiters_table.setItem(i, 5, QTableWidgetItem(str(recruiter['access_code'])))
        else:
            QMessageBox.warning(self, "Ошибка", "Не удалось загрузить данные рекрутеров")

```

```

def update_recruiter_info(self, row, fio, age, city, access_code, phone_number):
    success, message = get_recruiters_info(row, fio, age, city, access_code, phone_number)
    if success:
        self.recruiters_table.setItem(row, 1, QTableWidgetItem(fio))
        self.recruiters_table.setItem(row, 2, QTableWidgetItem(phone_number))
        self.recruiters_table.setItem(row, 3, QTableWidgetItem(age))
        self.recruiters_table.setItem(row, 4, QTableWidgetItem(city))
        self.recruiters_table.setItem(row, 5, QTableWidgetItem(access_code))
    else:
        QMessageBox.warning(self, "Ошибка", message)

```

Файл «server.py».

```

import psycopg2
connection = psycopg2.connect(user="postgres",
                               password="password1",
                               host="127.0.0.1",
                               port="5432",
                               database="candidate_testing_db")
cursor = connection.cursor()
cursor.execute('SET search_path TO main_schema,public')

def add_candidate(fio, age, city, access_code, phone_number):
    try:
        # Находим максимальный существующий номер кандидата и увеличиваем на 1
        cursor.execute('SELECT MAX("ID_candidate") FROM candidates')
        max_number = cursor.fetchone()[0]
        new_number = 1 if max_number is None else max_number + 1

        # SQL запрос для вставки новой записи
        insert_query = """
            INSERT INTO candidates ("ID_candidate", fio, age, city, access_code, phone_number)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        record_to_insert = (new_number, fio, age, city, access_code, phone_number)

        # Выполняем запрос
        cursor.execute(insert_query, record_to_insert)
        connection.commit()
        return True, access_code

    except (Exception, psycopg2.Error) as error:
        connection.rollback()
        return False, str(error)

def add_recruiter(fio, age, city, access_code, phone_number):
    try:
        # Находим максимальный существующий номер рекрутера и увеличиваем на 1
        cursor.execute('SELECT MAX("ID_recruiter") FROM recruiters')

```



```

max_number = cursor.fetchone()[0]
new_number = 1 if max_number is None else max_number + 1

# SQL запрос для вставки новой записи
insert_query = """
    INSERT INTO recruiters ("ID_recruiter", fio, age, city, access_code, phone_number)
    VALUES (%s, %s, %s, %s, %s, %s)
    """

record_to_insert = (new_number, fio, age, city, access_code, phone_number)

# Выполняем запрос
cursor.execute(insert_query, record_to_insert)
connection.commit()
return True, access_code

except (Exception, psycopg2.Error) as error:
    connection.rollback()
    return False, str(error)

def get_candidate_id_by_access_code(access_code):
    try:
        # SQL запрос для поиска ID кандидата по коду доступа
        select_query = """
            SELECT "ID_candidate"
            FROM candidates
            WHERE access_code = %s
            """

        # Выполняем запрос
        cursor.execute(select_query, (access_code,))
        result = cursor.fetchone()

        # Если найден кандидат, возвращаем его ID
        if result:
            return True, result[0]
        else:
            return False, "Кандидат с таким кодом доступа не найден"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

def get_recruiter_id_by_access_code(access_code):
    try:
        # SQL запрос для поиска ID рекрутера по коду доступа
        select_query = """
            SELECT "ID_recruiter"
            FROM recruiters
            WHERE access_code = %s
            """

        # Выполняем запрос
        cursor.execute(select_query, (access_code,))

```

```

result = cursor.fetchone()

# Если найден рекрутер, возвращаем его ID
if result:
    return True, result[0]
else:
    return False, "Рекрутер с таким кодом доступа не найден"

except (Exception, psycopg2.Error) as error:
    return False, str(error)

def get_candidate_fio_by_id(id_candidate):
    try:
        # SQL запрос для поиска ФИО кандидата по ID
        select_query = """
        SELECT fio
        FROM candidates
        WHERE "ID_candidate" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        result = cursor.fetchone()

        # Если найден кандидат, возвращаем его ФИО
        if result:
            return True, result[0]
        else:
            return False, "Кандидат с таким ID не найден"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

def get_recruiter_fio_by_id(id_recruiter):
    try:
        # SQL запрос для поиска ФИО рекрутера по ID
        select_query = """
        SELECT fio
        FROM recruiters
        WHERE "ID_recruiter" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_recruiter,))
        result = cursor.fetchone()

        # Если найден рекрутер, возвращаем его ФИО
        if result:
            return True, result[0]
        else:
            return False, "Рекрутер с таким ID не найден"

```

```

except (Exception, psycopg2.Error) as error:
    return False, str(error)

def get_candidates_count():
    try:
        # SQL запрос для подсчета всех записей в таблице candidates
        select_query = """
            SELECT COUNT(*)
            FROM candidates
        """

        # Выполняем запрос
        cursor.execute(select_query)
        result = cursor.fetchone()

        # Возвращаем количество записей
        if result:
            return True, result[0]
        else:
            return False, "Не удалось получить количество кандидатов"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

def delete_candidate_by_id(id_candidate):
    try:
        # SQL запросы для удаления ответов кандидата из всех таблиц с ответами
        delete_answers_queries = [
            """DELETE FROM candidates_answers_theory1 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory2 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory3 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_theory4 WHERE "ID_candidate" = %s""",
            """DELETE FROM candidates_answers_logic WHERE "ID_candidate" = %s"""
        ]

        # Удаляем ответы кандидата из всех таблиц
        for query in delete_answers_queries:
            cursor.execute(query, (id_candidate,))

        # SQL запрос для удаления кандидата по ID
        delete_query = """
            DELETE FROM candidates
            WHERE "ID_candidate" = %s
        """

        # Выполняем запрос на удаление кандидата
        cursor.execute(delete_query, (id_candidate,))

        # Подтверждаем изменения
        connection.commit()

        # Проверяем, была ли удалена запись

```

```

        if cursor.rowcount > 0:
            return True, "Кандидат успешно удален"
        else:
            return False, "Кандидат с таким ID не найден"

except (Exception, psycopg2.Error) as error:
    # Откатываем изменения в случае ошибки
    connection.rollback()
    return False, str(error)

def delete_recruiter_by_id(id_recruiter):
    try:
        # SQL запрос для удаления рекрутера по ID
        delete_query = """
            DELETE FROM recruiters
            WHERE "ID_recruiter" = %s
        """

        # Выполняем запрос на удаление рекрутера
        cursor.execute(delete_query, (id_recruiter,))

        # Подтверждаем изменения
        connection.commit()

        # Проверяем, была ли удалена запись
        if cursor.rowcount > 0:
            return True, "Рекрутер успешно удален"
        else:
            return False, "Рекрутер с таким ID не найден"

except (Exception, psycopg2.Error) as error:
    # Откатываем изменения в случае ошибки
    connection.rollback()
    return False, str(error)

def get_candidate_info_by_id(id_candidate):
    try:
        # SQL запрос для поиска информации о кандидате по ID
        select_query = """
            SELECT fio, city, phone_number
            FROM candidates
            WHERE "ID_candidate" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        result = cursor.fetchone()

        # Если найден кандидат, возвращаем его данные
        if result:
            return True, {
                'fio': result[0],

```

```

        'city': result[1],
        'number': result[2]
    }
    else:
        return False, "Кандидат с таким ID не найден"

except (Exception, psycopg2.Error) as error:
    return False, str(error)

def check_answers_exist_theory1(id_candidate):
    try:
        # SQL запрос для проверки наличия записей для данного кандидата
        select_query = """
            SELECT COUNT(*)
            FROM candidates_answers_theory1
            WHERE "ID_candidate" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        result = cursor.fetchone()

        # Если количество записей больше 0, возвращаем True
        if result and result[0] > 0:
            return True, "Записи найдены"
        else:
            return False, "Записи не найдены"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

```

(Для проверки прохождения других тестов пользователем реализованы аналогичные функции.)

```

def get_recruiters_info():
    try:
        # SQL запрос для получения ФИО и телефона всех рекрутеров
        select_query = """
            SELECT fio, phone_number, age, city, access_code
            FROM recruiters
            ORDER BY "ID_recruiter"
        """

        # Выполняем запрос
        cursor.execute(select_query)
        results = cursor.fetchall()

        # Если найдены рекрутеры, возвращаем список их данных
        if results:
            recruiters = [{'fio': row[0], 'phone_number': row[1], 'age': row[2], 'city': row[3],

```

```

'access_code': row[4]} for row in results]
    return True, recruiters
else:
    return False, "Рекрутеры не найдены"

except (Exception, psycopg2.Error) as error:
    return False, str(error)

def get_recruiters_count():
    try:
        # SQL запрос для подсчета количества рекрутеров
        select_query = """
            SELECT COUNT(*)
            FROM recruiters
        """

        # Выполняем запрос
        cursor.execute(select_query)
        result = cursor.fetchone()

        # Возвращаем количество рекрутеров
        if result:
            return True, result[0]
        else:
            return False, "Не удалось получить количество рекрутеров"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

def get_answers_list_by_candidate_theory1(id_candidate):
    try:
        # SQL запрос для получения ответов кандидата
        select_query = """
            SELECT answer
            FROM candidates_answers_theory1
            WHERE "ID_candidate" = %s
            ORDER BY "ID_question"
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_candidate,))
        results = cursor.fetchall()

        # Преобразуем результаты в список чисел
        if results:
            answers = [row[0] for row in results]
            return True, answers
        else:
            return False, "Ответы не найдены"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)

```

(Для получения списка ответов кандидата по другим тестам реализованы аналогичные функции)

```
def get_question_by_id_theory1(id_question):
    try:
        # SQL запрос для получения вопроса и вариантов ответа по ID
        select_query = """
            SELECT question, answer1, answer2, answer3, answer4, correct_answer
            FROM py_theory1
            WHERE "ID_question" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_question,))
        result = cursor.fetchone()

        # Если найден вопрос, возвращаем его и варианты ответов
        if result:
            return True, {
                'question': result[0],
                'answer1': result[1],
                'answer2': result[2],
                'answer3': result[3],
                'answer4': result[4],
                'correct_answer': result[5]
            }
        else:
            return False, "Вопрос с таким ID не найден"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)
```

(Для получения вопроса и вариантов ответа по его ID других тестов реализованы аналогичные функции.)

```
def save_answer(id_candidate, id_question, answer):
    try:
        # SQL запрос для вставки ответа кандидата
        insert_query = """
            INSERT INTO candidates_answers_theory1 ("ID_candidate", "ID_question", answer)
            VALUES (%s, %s, %s)
        """

        # Выполняем запрос
        cursor.execute(insert_query, (id_candidate, id_question, answer))
        connection.commit()

        return True, "Ответ успешно сохранен"
```

```
except (Exception, psycopg2.Error) as error:  
    return False, str(error)
```

(Для сохранения ответов по другим тестам реализованы аналогичные функции.)

```
def get_answers_list_theory1(id_candidate):  
    try:  
        # SQL запрос для получения ответов кандидата  
        select_query = """  
            SELECT "ID_question", answer  
            FROM candidates_answers_theory1  
            WHERE "ID_candidate" = %s  
            ORDER BY "ID_question"  
        """  
  
        # Выполняем запрос  
        cursor.execute(select_query, (id_candidate,))  
        results = cursor.fetchall()  
  
        # Преобразуем результаты в список кортежей  
        answers = [(row[0], row[1]) for row in results]  
  
        return True, answers  
  
    except (Exception, psycopg2.Error) as error:  
        return False, str(error)
```

(Для получения ответов кандидата в виде списка кортежей по другим тестам реализованы аналогичные функции.)

```
def get_question_text_by_id_theory1(id_question):  
    try:  
        # SQL запрос для получения текста вопроса по ID  
        select_query = """  
            SELECT question  
            FROM py_theory1  
            WHERE "ID_question" = %s  
        """  
  
        # Выполняем запрос  
        cursor.execute(select_query, (id_question,))  
        result = cursor.fetchone()  
  
        # Если найден вопрос, возвращаем его текст  
        if result:  
            return True, result[0]  
        else:  
            return False, "Вопрос с таким ID не найден"
```



```
except (Exception, psycopg2.Error) as error:
    return False, str(error)
```

(Для получения текста вопроса по его ID других тестов реализованы аналогичные функции.)

```
def get_answer_by_id_theory1(id_question, answer_number):
    try:
        # Проверяем корректность номера ответа
        if answer_number not in [1, 2, 3, 4]:
            return False, "Некорректный номер ответа. Допустимые значения: 1, 2, 3, 4"

        # SQL запрос для получения конкретного варианта ответа по ID вопроса
        select_query = f"""
            SELECT answer{answer_number}
            FROM py_theory1
            WHERE "ID_question" = %s
        """

        # Выполняем запрос
        cursor.execute(select_query, (id_question,))
        result = cursor.fetchone()

        # Если найден ответ, возвращаем его
        if result:
            return True, result[0]
        else:
            return False, "Вопрос с таким ID не найден"

    except (Exception, psycopg2.Error) as error:
        return False, str(error)
```

(Для получения варианта ответа пользователя на вопросы в других тестах реализованы аналогичные функции.)

```
def get_answers_listt_theory1():
    try:
        # SQL запрос для получения всех правильных ответов из таблицы py_theory1
        select_query = """
            SELECT correct_answer
            FROM py_theory1
            ORDER BY "ID_question"
        """

        # Выполняем запрос
        cursor.execute(select_query)
        results = cursor.fetchall()

        # Преобразуем результаты в список
        correct_answers = [result[0] for result in results]
```

```
return True, correct_answers
```

```
except (Exception, psycopg2.Error) as error:  
    return False, str(error)
```

(Для получения правильных ответов по другим тестам реализованы аналогичные функции.)

```
def reorder_candidate_ids():  
    try:  
        # Получаем все ID кандидатов и сортируем их  
        select_query = """  
            SELECT "ID_candidate"  
            FROM candidates  
            ORDER BY "ID_candidate"  
        """  
        cursor.execute(select_query)  
        old_ids = [row[0] for row in cursor.fetchall()]  
  
        # Создаем новый список ID (1, 2, 3, ...)  
        new_ids = list(range(1, len(old_ids) + 1))  
  
        # Обновляем ID в таблице candidates  
        for old_id, new_id in zip(old_ids, new_ids):  
            if old_id != new_id:  
                # Обновляем ID в таблице candidates  
                update_candidate_query = """  
                    UPDATE candidates  
                    SET "ID_candidate" = %s  
                    WHERE "ID_candidate" = %s  
                """  
                cursor.execute(update_candidate_query, (new_id, old_id))  
  
        # Обновляем ID в связанных таблицах  
        tables = [  
            "candidates_answers_theory1",  
            "candidates_answers_theory2",  
            "candidates_answers_theory3",  
            "candidates_answers_theory4",  
            "candidates_answers_logic"  
        ]  
  
        for table in tables:  
            update_answers_query = f"""  
                UPDATE {table}  
                SET "ID_candidate" = %s  
                WHERE "ID_candidate" = %s  
            """  
            cursor.execute(update_answers_query, (new_id, old_id))
```

```

# Подтверждаем изменения
connection.commit()
return True, "ID кандидатов успешно перенумерованы"

except (Exception, psycopg2.Error) as error:
    # Откатываем изменения в случае ошибки
    connection.rollback()
    return False, str(error)

```

Файл «*analis.py*».

```

from server import (get_answers_list_theory1, get_answers_list_theory1, get_answers_list_theory2,
                    get_answers_list_theory2, get_answers_list_theory3, get_answers_list_theory3,
                    get_answers_list_theory4, get_answers_list_theory4, get_answers_list_logic,
                    get_answers_list_logic)
from priorities import get_priorities as get_priorities_from_module

def get_correct_answers_count_theory1(id_candidate):
    try:
        # Получаем ответы кандидата
        success, candidate_answers = get_answers_list_theory1(id_candidate)
        if not success:
            return False, candidate_answers # Возвращаем ошибку

        # Получаем правильные ответы
        success, correct_answers = get_answers_list_theory1()
        if not success:
            return False, correct_answers # Возвращаем ошибку

        # Считаем количество правильных ответов
        correct_count = 0
        for candidate_answer, correct_answer in zip(candidate_answers, correct_answers):
            if candidate_answer[1] == correct_answer: # candidate_answer[1] - это ответ кандидата
                correct_count += 1

        return True, correct_count

    except Exception as error:
        return False, str(error)

```

(Для подсчёта правильных ответов кандидата по другим тестам реализованы аналогичные функции.)

```

def get_priorities():
    try:
        # Получаем приоритеты из модуля priorities
        return get_priorities_from_module()
    except Exception as error:
        return False, str(error)

```

```

def geometric_mean(numbers):
    product = 1
    for num in numbers:
        product *= num
    return product ** (1/len(numbers))

def final_priorities(vector):
    sum_vector = sum(vector)
    for i in range(len(vector)):
        vector[i] = vector[i] / sum_vector
    return vector

def get_candidate_score(id_candidate):
    try:
        print(f"\nРасчет score для кандидата {id_candidate}:")

        priorities = get_priorities()
        print(f"Полученные приоритеты: {priorities}")

        if isinstance(priorities, tuple) and not priorities[0]:
            return False, priorities[1]

        vector_priorities = []
        for row in priorities:
            mean = geometric_mean(row)
            print(f"Геометрическое среднее для строки {row}: {mean}")
            vector_priorities.append(mean)

        vector_priorities = final_priorities(vector_priorities)
        print(f"Нормализованные приоритеты: {vector_priorities}")

        # Получаем результаты тестов
        test_results = [
            get_correct_answers_count_theory1(id_candidate),
            get_correct_answers_count_theory2(id_candidate),
            get_correct_answers_count_theory3(id_candidate),
            get_correct_answers_count_theory4(id_candidate),
            get_correct_answers_count_logic(id_candidate)
        ]

        print(f"Результаты тестов: {test_results}")

        # Проверяем успешность получения ответов и извлекаем количество правильных
        # ответов
        correct_answers = []
        for success, count in test_results:
            if not success:
                print(f"Ошибка при получении результатов теста: {count}")
                return False, count
            correct_answers.append(count)

```

```

print(f'Количество правильных ответов: {correct_answers}')

# Вычисляем итоговый score
score = 0
for i in range(5):
    score += vector_priorities[i] * correct_answers[i]
    print(f'Вклад теста {i+1}: {vector_priorities[i]} * {correct_answers[i]} = {vector_priorities[i] * correct_answers[i]}')

print(f'Итоговый score: {score}')
return True, score
except Exception as error:
    print(f'Ошибка при расчете score: {str(error)}')
    return False, str(error)

```

Файл «properties.py».

```

# Глобальная переменная для хранения приоритетов
priorities = [[1.0 for _ in range(5)] for _ in range(5)]

def get_priorities():
    return priorities

def set_priorities(new_priorities):
    global priorities
    priorities = new_priorities

```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ОТЗЫВ РУКОВОДИТЕЛЯ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

на тему Моделирование процессов подбора и оценки персонала
в кадровом менеджменте

выполненную студентом группы № М112

Поспеловой Анастасией Александровной

фамилия, имя, отчество студента

по направлению подготовки/ 01.03.02 Прикладная математика и информатика
специальности (код) (наименование направления подготовки/специальности)

(наименование направления подготовки/специальности)

Актуальность темы работы: в настоящее время большинство организаций для определения качеств кандидатов на должность применяют трудоемкие традиционные методы без использования средств автоматизации. В результате работа по подбору персонала занимает значительное время, которое уходит на выявление уровня профессиональной подготовки кандидатов на основе прохождения контрольно-измерительных испытаний, а затем на обработку данных. Таким образом, процесс выбора наиболее подходящего сотрудника усложняется. Для более оперативного определения подходящего кандидата на соответствующую должность необходимо разрабатывать и применять современные методы с использованием средств автоматизации, что обуславливает актуальность темы работы.

Цель работы: повышение эффективности подбора персонала путем автоматизации процесса оценивания кандидатов на должность

Задачи исследования: провести анализ существующих подходов к подбору и оценке персонала в кадровом менеджменте; предложить методы, позволяющие повысить эффективность процесса подбора кандидата на должность; разработать информационную систему оценки кандидатов на должность в IT-индустрии.

Общая оценка выполнения поставленной перед студентом задачи, основные достоинства и недостатки работы:

Цель, поставленная перед А.А. Поспеловой, достигнута, задачи выполнены. Структура работы логична и соответствует поставленным задачам.

Степень самостоятельности и способности к исследовательской работе студент:

При написании выпускной квалификационной работы А.А. Поспелова продемонстрировала полученные за время обучения в университете знания и навыки самостоятельной работы, умение применять их для решения прикладных задач.

Проверка текста выпускной квалификационной работы с использованием системы

___АНТИПЛАГИАТ.ВУЗ___, проводившаяся «_07_» _июня_ 2025 г., показывает оригинальность содержания на уровне _95,22_ %.

В работе не содержится информация с ограниченным доступом, и отсутствуют сведения, представляющие коммерческую ценность.

Степень грамотности изложения и оформления материала:

Изложение и оформление материала соответствует «Положению о выпускной квалификационной работе».

Оценка деятельности студента в период подготовки выпускной квалификационной работы:

В период подготовки выпускной квалификационной работы А.А. Поспелова проявила себя работоспособной, дисциплинированной и ответственной студенткой.


Общий вывод:

Задание по выпускной квалификационной работе А.А. Поспелова выполнила. Выпускная квалификационная работа соответствует предъявляемым требованиям, а А.А. Поспеловой может быть присвоена квалификация «Бакалавр» по направлению 01.03.02 «Прикладная математика и информатика».

Руководитель

профессор, д.т.н., доцент

должность, уч. степень, звание

 10.06.2025

подпись, дата

Л. П. Вершинина

инициалы, фамилия



ГУАП

Моделирование процессов подбора и оценки персонала в кадровом менеджменте

Исполнитель: студентка гр. М112 А.А. Поспелова

Научный руководитель: профессор, д.т.н., доцент Л.П. Вершинина

2025

Цель исследования: повышение эффективности подбора персонала путём оптимизации и автоматизации процесса оценивания кандидатов на должность.

Объект исследования: процесс подбора и оценки персонала в компании.

Задачи исследования:

- провести анализ существующих подходов к подбору и оценке персонала в кадровом менеджменте и выявить проблемы;
- предложить методы, позволяющие повысить эффективность процесса подбора кандидата на должность;
- разработать информационную систему оценки кандидатов на должность в IT-индустрии.

Предмет исследования: методы принятия решения при подборе персонала.

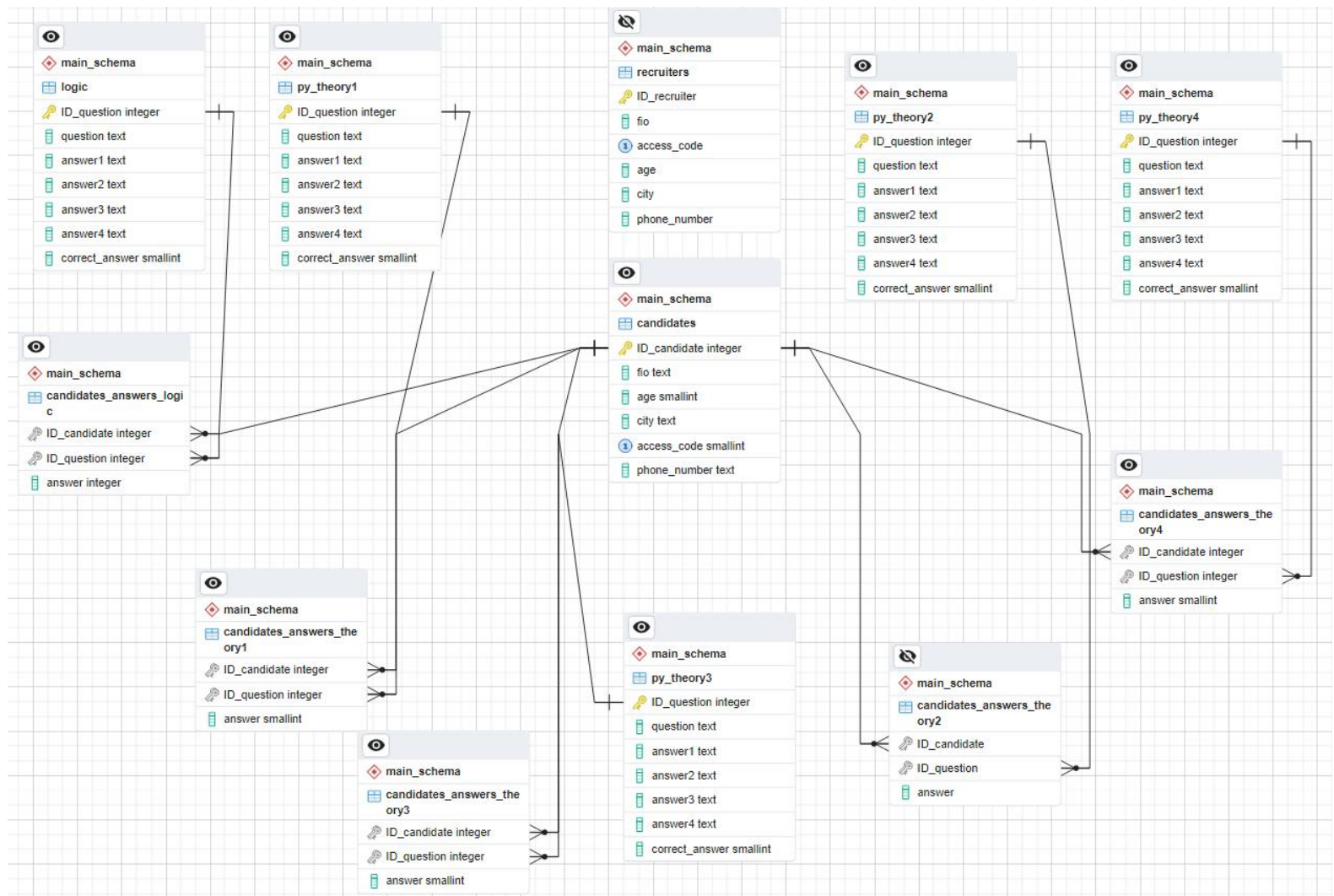


Схема базы данных

Добро пожаловать

Добро пожаловать

Регистрация

Введите код авторизации:

Введите ваш код доступа

Авторизоваться

Меню

Главная Тесты Ответы

Добро пожаловать, Никольский Платон Алексеевич!

Авторизация и приветственное меню

Тест 3

Какой метод переноса виртуального окружения считается наиболее надежным для обеспечения воспроизводимости проекта?

☐

 Прямое копирование директории виртуального окружения (zip/tar)

☐

 Заморозка зависимостей в requirements.txt и установка на новой машине

☐

 Экспорт списка пакетов через pip list и ручная установка

☐

 Копирование всего жесткого диска с проектом

Следующий вопрос

Результаты теста 4

| | Вопрос | Ответ |
|----|--|---|
| 5 | Что представляет собой объект Task в asyncio? | Запланированная корутина для выполнения |
| 6 | Что делает функция asyncio.gather(*tasks)? | Запускает все задачи параллельно и дожидается завершения всех задач |
| 7 | Какой из приведенных вариантов НЕ является преимуществом asyncio? | Подходит для всех типов задач, включая CPU-bound |
| 8 | Какую функцию выполняет asyncio.run(main())? | Создает новый event loop, запускает корутину main() и закрывает цикл после завершения |
| 9 | Какое утверждение является верным касательно использования asyncio с синхронными библиотеками? | Asyncio автоматически преобразует синхронные библиотеки в асинхронные |
| 10 | Для чего используется метод task.cancel() у объекта Task в asyncio? | Для перезапуска корутины с начала |

Закреть

Меню рекрутера

Добро пожаловать, Кулагин Филипп Леонидович!

Главная Кандидаты Пользователи Аналитика

| № | ФИО | Город | Телефон |
|---|-------------------------------|-----------------|--------------|
| 1 | Никольский Платон Алексеевич | Санкт-Петербург | 89134562365 |
| 2 | Тихонова Василиса Данииловна | Санкт-Петербург | 89511234567 |
| 3 | Михайлова Алиса Александровна | Санкт-Петербург | 891198777654 |
| 4 | Михайлова Александра ... | Москва | 89997655675 |
| 5 | Воробьев Андрей Вячеславович | Санкт-Петербург | 89112233322 |
| 6 | Тихомиров Артём Григорьевич | Санкт-Петербург | 89315678790 |

Меню рекрутера

Добро пожаловать, Кулагин Филипп Леонидович!

Главная Кандидаты Пользователи Аналитика

| № | ФИО | Телефон | Возраст | Город | Код доступа |
|---|--------------------|-------------|---------|-----------------|-------------|
| 1 | Кулагин Филипп ... | 89119752823 | 35 | Санкт-Петербург | 2345 |

Зарегистрированные кандидаты и рекрутеры

Шкала оценок решений

| Интенсивность важности a_{ij} | Качественная оценка | Объяснения |
|---------------------------------|--------------------------------------|--|
| 0 | Несравнимость | Нет смысла сравнивать решения |
| 1 | Одинаковая значимость | Решения равны по значимости |
| 3 | Слабо значимее | Существуют показания о предпочтении одного решения другому, но показания неубедительные |
| 5 | Существенно или сильно значимее | Существуют хорошие доказательства и логические критерии, которые могут показать, что элемент более важен |
| 7 | Очевидно значимее | Существует очевидное доказательство большей значимости одного элемента перед другим |
| 9 | Абсолютно значимее | Максимально подтверждается ощутимость предпочтения одного элемента другому |
| 2, 4, 6, 8 | Промежуточные оценки между соседними | |

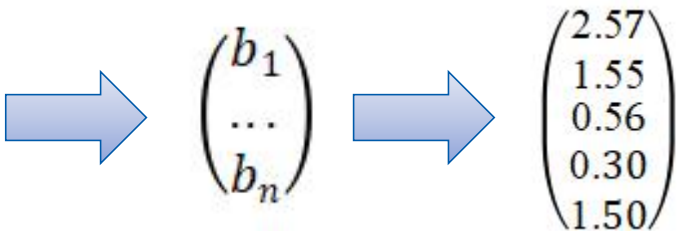
| | 1 | 2 | 3 | 4 | 5 |
|---|------|------|-----|-----|------|
| 1 | 1.0 | 2.0 | 4.0 | 7.0 | 2.0 |
| 2 | 0.5 | 1.0 | 3.0 | 6.0 | 1.0 |
| 3 | 0.25 | 0.33 | 1.0 | 2.0 | 0.33 |
| 4 | 0.14 | 0.17 | 0.5 | 1.0 | 0.2 |
| 5 | 0.5 | 1.0 | 3.0 | 5.0 | 1.0 |

Матрица парных сравнений

$$b_1 = \sqrt[n]{a_{11} * \dots * a_{1n}}$$

...

$$b_n = \sqrt[n]{a_{n1} * \dots * a_{nn}}$$



Вычисление локальных приоритетов

$$c_i = \frac{b_i}{b_1 + \dots + b_n}$$

$$c_1 = \frac{b_1}{b_1 + \dots + b_5} = \frac{2.57}{6.48} = 0.39 \quad \longrightarrow \quad \begin{pmatrix} 0.39 \\ 0.24 \\ 0.09 \\ 0.05 \\ 0.23 \end{pmatrix}$$

Нормализуем

$$C = \sum_{i=1}^n c_i * k_i$$

$$C_1 = \sum_{i=1}^5 c_i * k_i = 0.39 * 10 + 0.24 * 7 + 0.09 * 7 + 0.05 * 8 + 0.23 * 7 = 8.24$$

Получаем итоговые
оценки

Аналитика

Аналитика результатов

| | № | ФИО | Результат |
|---|---|--------------------------------|-----------|
| 1 | 1 | Никольский Платон Алексеевич | 8.24 |
| 2 | 2 | Тихонова Василиса Данииловна | 2.43 |
| 3 | 3 | Михайлова Алиса Александровна | 6.23 |
| 4 | 4 | Михайлова Александра Фёдоровна | 5.43 |
| 5 | 5 | Воробьев Андрей Вячеславович | 5.44 |
| 6 | 6 | Тихомиров Артём Григорьевич | 9.12 |

Рекомендуемый кандидат: Тихомиров Артём Григорьевич

Результат вычислений



Спасибо за внимание!