СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В текущих условиях банковская сфера занимает центральное место в экономике, обеспечивая не только финансовую стабильность, но и поддержку бизнес-процессов в различных отраслях. Эффективное управление человеческими ресурсами становится критически важным фактором для успеха организации, поскольку именно от квалификации и мотивации сотрудников зависит уровень обслуживания клиентов и конкурентоспособность банка. В условиях глобализации и быстрого технологического прогресса банки сталкиваются с необходимостью адаптации своих внутренних процессов, особенно в области управления персоналом.

Актуальность. В условиях быстрого развития банковских технологий и растущих требований клиентов автоматизация управления персоналом становится необходимостью для повышения качества обслуживания и снижения ошибок, связанных с человеческим фактором. Современные банки должны адаптироваться к изменениям на рынке, внедряя инновационные подходы и технологии для эффективного управления кадрами. Автоматизация процессов учёта рабочего времени и расчёта заработной платы позволит значительно сократить время обработки данных и повысить точность отчётности.

Объектом данного проекта являются процессы управления персоналом в банковской сфере, в то время как предметом является система автоматизации управления этими процессами. Целью курсового проектирования является улучшение существующих процессов управления персоналом путём создания автоматизированной системы управления персоналом в банковской сфере.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– провести исследование существующих автоматизированных систем управления персоналом в банковской сфере для выявления ключевых требований, разработать функциональную модель процессов управления и спроектировать информационную структуру;

– разработать архитектуру и дизайн системы автоматизации управления персоналом, включая выбор технологий и инструментов разработки;

– провести комплексное тестирование системы, включая функциональное, производительное и тестирование безопасности, чтобы гарантировать её надёжность и отсутствие ошибок;

– подготовить подробное руководство для пользователей, объясняющее, как развернуть и эффективно использовать систему управления персоналом в банке.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ПЕРСОНАЛОМ В БАНКОВСКОЙ СФЕРЕ

1.1 Описание автоматизации управления персоналом в банковской сфере

Система управления персоналом в банковской сфере представляет собой ключевой инструмент для повышения эффективности работы финансовых учреждений. В условиях высокой конкуренции и постоянных изменений на рынке, обеспечивающих стабильность и рост, необходима автоматизация процессов, связанных с учетом рабочего времени, расчетом заработной платы и ведением кадрового делопроизводства. Проблема управления персоналом в банках становится все более актуальной, поскольку многие учреждения сталкиваются с неэффективностью существующих процессов, которые зачастую выполняются вручную или с помощью устаревших программных решений. Это приводит к задержкам в обработке данных, ошибкам в начислениях и затрудняет доступность актуальной информации для всех заинтересованных сторон. В результате сотрудники и руководители теряют время на выполнение рутинных задач, что негативно сказывается на качестве обслуживания клиентов и общем уровне эффективности работы банка.

Автоматизация управления персоналом имеет множество преимуществ, включая сокращение времени на обработку данных, что обеспечивает оперативность в принятии решений, а также минимизацию человеческих ошибок, что особенно важно в финансовой сфере. Внедрение системы учета рабочего времени и расчета заработной платы позволяет создать прозрачную и доступную для анализа отчетность, что способствует более эффективному управлению ресурсами банка. Основные процессы, которые будут автоматизированы, включают сбор данных о сотрудниках, учет рабочего времени, расчет заработной платы и формирование отчетов по начислениям. Объектами автоматизации являются личные дела сотрудников, табели учета рабочего времени и данные о начислениях и вычетах.

Важно учитывать потребности различных заинтересованных лиц, таких как сотрудники отдела кадров, руководители подразделений, сотрудники банка, соискатели и администраторы системы. Каждая из этих групп имеет свои уникальные требования к функциональности системы. Например, сотрудники отдела кадров нуждаются в удобном интерфейсе для учета рабочего времени и расчетов, в то время как руководители требуют отчетности по производительности сотрудников. Сотрудники хотят иметь доступ к информации о своих начислениях, а администраторам требуется функционал для управления профилями пользователей.

Для реализации функций по расчёту заработной платы в системе можно использовать различные способы. В практике предприятий применяются

различные формы оплаты труда, такие как повременная и сдельная, каждая из которых имеет свои особенности и подходит для разных условий работы. Повременная форма оплаты основана на начислении заработной платы в зависимости от фактически отработанного времени. Она включает в себя простую повременную систему, повременно-премиальную и окладную. Простая повременная система предполагает расчет заработной платы на основе отработанных часов, тогда как повременно-премиальная система добавляет элемент премирования за выполнение качественных и количественных показателей. Окладная система фиксирует заработную плату по установленным должностным окладам, что обеспечивает стабильность дохода для работников.

С другой стороны, сдельная форма оплаты труда основывается на количестве и качестве выполненных работ или произведенной продукции. Существуют различные системы сдельной оплаты, включая прямую сдельную, сдельно-премиальную, сдельно-прогрессивную и косвенно-сдельную, каждая из которых предоставляет различные механизмы стимулирования работников в зависимости от их производительности. Например, сдельно-премиальная система позволяет работнику получать премии за перевыполнение установленных норм, что может способствовать повышению мотивации.

В банковской сфере более предпочтительными являются повременные формы оплаты труда. Данные формы обеспечивают стабильность и предсказуемость дохода, что важно для сотрудников, работающих в условиях высокой ответственности. Повременно-премиальная система может дополнительно мотивировать сотрудников, улучшая их производительность и качество работы. Использование этих систем также способствует созданию более устойчивой и эффективной рабочей среды, что критически важно в финансовом секторе.

Обзор существующих аналогов показывает, что на рынке представлены различные решения для автоматизации управления персоналом, среди которых можно выделить SAP SuccessFactors, 1С:Зарплата и управление персоналом, а также Oracle HCM Cloud. Каждое из этих решений обладает уникальными достоинствами и недостатками, которые следует учитывать при выборе подходящей системы.

SAP SuccessFactors Employee Central — это гибкая облачная информационная система для управления персоналом (HRIS), призванная помочь вам в управлении бизнесом и кадровым составом [1]. SAP SuccessFactors предлагает широкий спектр функций для управления персоналом, включая управление талантами, обучение, оценку производительности и анализ данных. Это решение позволяет интегрировать различные HR-процессы в единую систему, что делает его удобным для крупных организаций со сложными структурами. Однако, несмотря на свои преимущества, SAP SuccessFactors имеет высокую стоимость внедрения и обслуживания, что может стать значительным

барьером для многих компаний. Кроме того, его интерфейс может быть сложным для пользователей, требуя дополнительного обучения и времени на адаптацию.

"1С:Зарплата и управление персоналом" — программа, позволяющая в комплексе автоматизировать задачи, связанные с расчетом заработной платы персонала и реализацией кадровой политики, с учетом требований законодательства Республики Беларусь и реальной практики работы предприятий [2]. Она является популярным решением на постсоветском пространстве благодаря своей доступности и простоте использования. Это программное обеспечение предлагает основные функции для учета рабочего времени и расчета заработной платы, что делает его привлекательным для малых и средних предприятий. Однако его возможности интеграции с другими системами ограничены, что может стать проблемой для организаций, стремящихся создать более сложные экосистемы управления. Упрощенный функционал может не удовлетворять потребности крупных банков или организаций с высокими требованиями к автоматизации.

Oracle Fusion Cloud HCM — это комплексное облачное решение, которое объединяет все процессы управления персоналом и каждого человека на предприятии [3]. Oracle HCM Cloud представляет собой мощное решение, которое предлагает высокую степень автоматизации и аналитики. Это программное обеспечение включает инструменты для управления талантами, анализа производительности и формирования отчетности, что позволяет организациям принимать обоснованные решения на основе данных. Тем не менее, Oracle HCM Cloud также имеет высокую стоимость, как внедрения, так и поддержки. Процесс внедрения может быть длительным и сложным, что требует значительных временных и финансовых ресурсов. Это может быть серьезным ограничением для организаций, которые хотят быстро адаптироваться к новым условиям или ограничены в бюджете.

Таким образом, разработка системы управления персоналом в банковской сфере позволит повысить качество обслуживания клиентов, оптимизировать внутренние процессы и увеличить общую эффективность работы банка, что делает данную систему стратегически важной задачей для финансовых учреждений.

1.2 Разработка функциональной модели предметной области

Для разработки функциональной модели использовалась методология IDEF0. IDEF0 – Function Modeling – методология функционального моделирования, позволяющая описать процесс в виде иерархической системы взаимосвязанных функций [4].

Основным блоком контекстной диаграммы является процесс “Начислить заработную плату”. Данный блок изображён на рисунке 1.1. Входными дугами являются следующие дуги: “Общие данные о сотрудниках”, которые содержат

информацию, необходимую для расчета заработной платы, включая личные данные, должности и оклады сотрудников и “Данные о начислениях и вычетах”, включающие информацию о дополнительных выплатах, премиях, налогах и прочих вычетах, которые должны быть учтены при расчете. В результате выполнения процесса формируется выходная дуга — Архивированный отчёт по начислению заработной платы, который содержит все необходимые данные о начислениях для каждого сотрудника и может быть использован для дальнейшей отчетности. Осуществляется процесс при помощи сотрудника отдела кадров и системы управления персоналом.

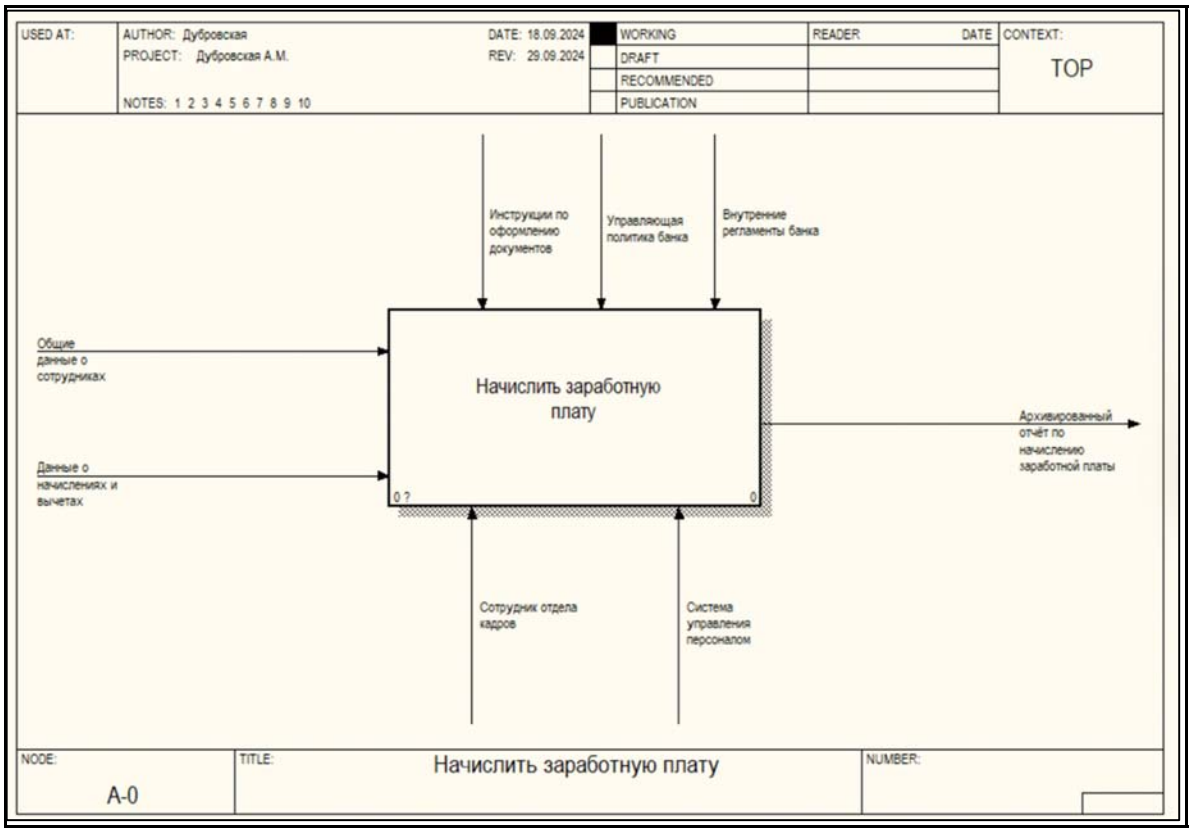


Рисунок 1.1 - Контекстная диаграмма

На рисунке 1.2 изображена декомпозиция главного блока, которая состоит из трёх процессов:

– собрать данные для расчёта заработной платы;

– рассчитать заработную плату;

– оформить начисление заработной платы.

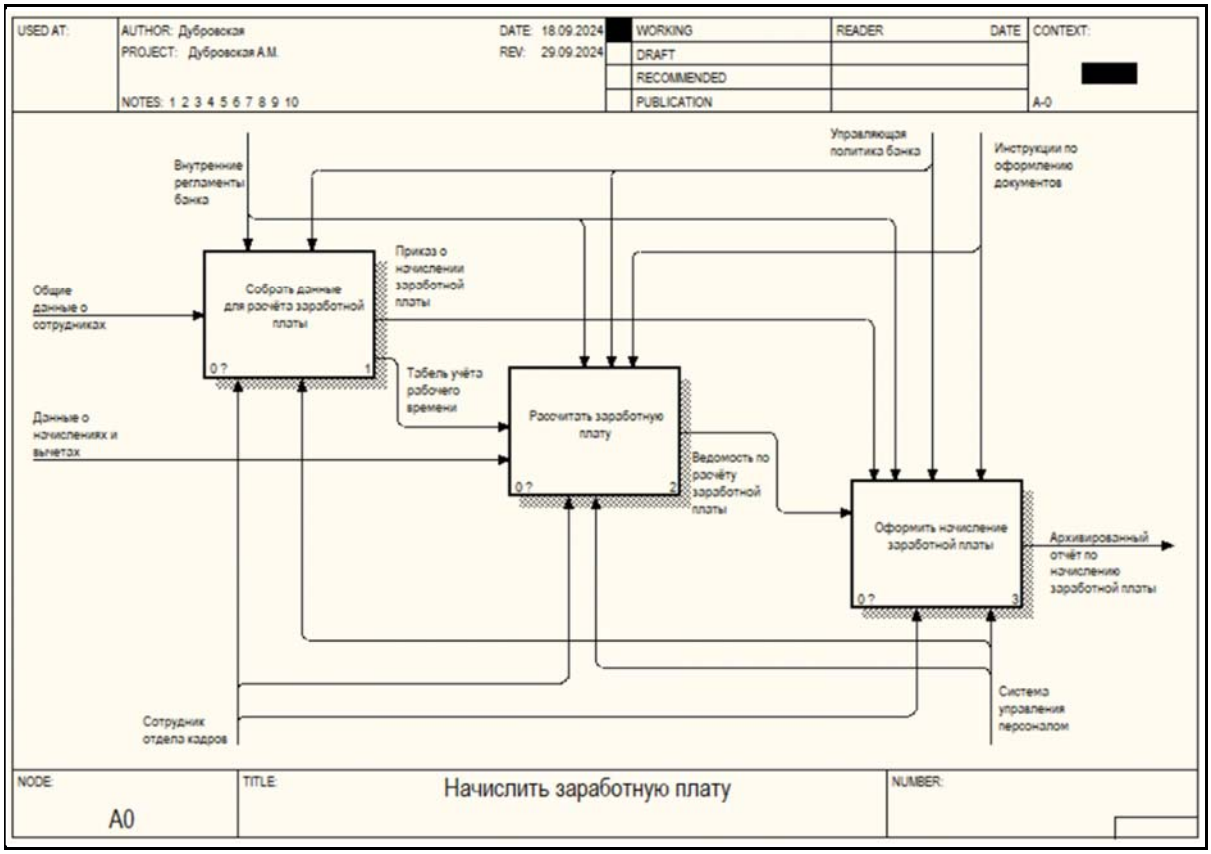


Рисунок 1.2 - Декомпозиция блока “Начислить заработную плату”

На рисунке 1.3 изображена декомпозиция блока “Собрать данные для расчёта заработной платы”. Первым является процесс “Актуализировать данные о сотрудниках”, итог которого - актуализированные данные о сотрудниках. Этот процесс включает в себя получение информации о новых сотрудниках, обновление данных действующих сотрудников, проверку точности первичных данных (см. рисунок 1.4). Вторым является процесс “Собрать информацию о времени работы”, который состоит из следующих процессов: собрать табели учёта рабочего времени, обработать данные о фактически отработанном времени, проверить и утвердить табели (см. рисунок 1.5). Далее выполняется процесс “Подготовить приказ о начислении заработной платы”, итогом этого процесса является приказ о начислении заработной платы. Для подготовки приказа о начислении зарплаты необходимо составить список сотрудников для начисления, подготовить необходимые документы, убедиться в актуальности документов (см. рисунок 1.6).

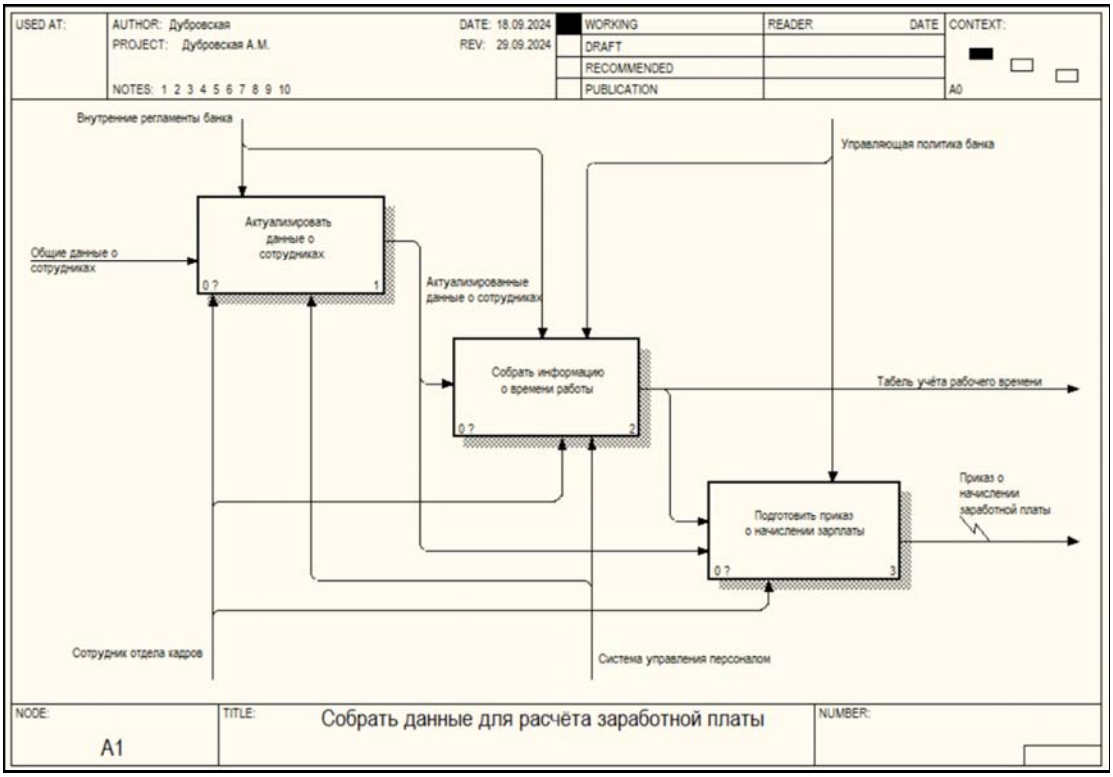


Рисунок 1.3 - Декомпозиция блока “Собрать данные для расчёта заработной платы”

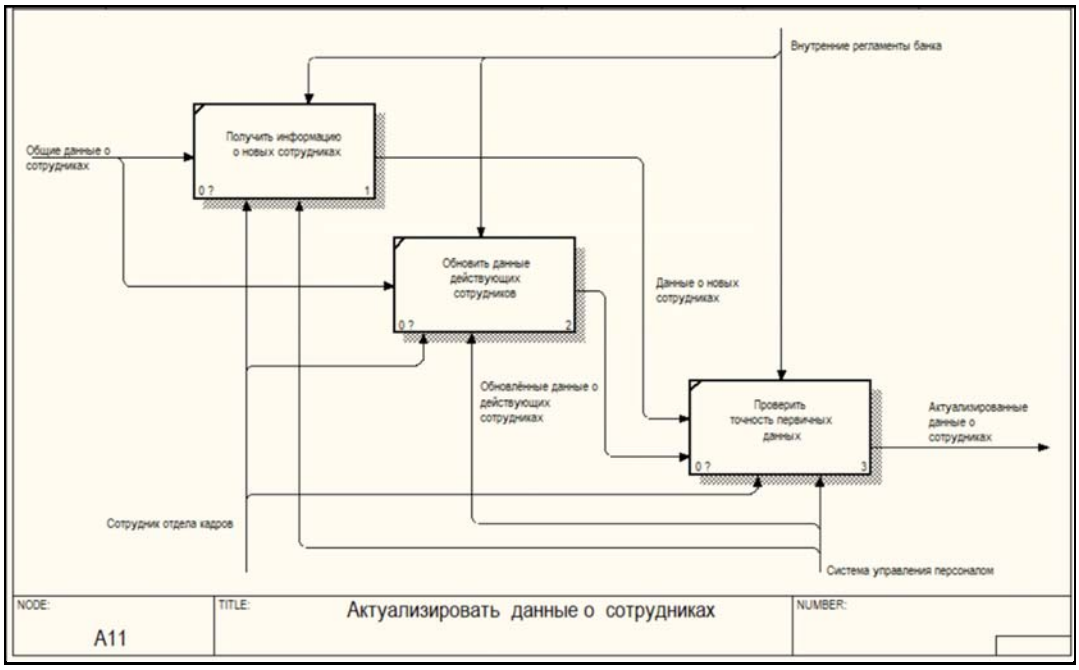


Рисунок 1.4 – Декомпозиция блока “Актуализировать данные о сотрудниках”

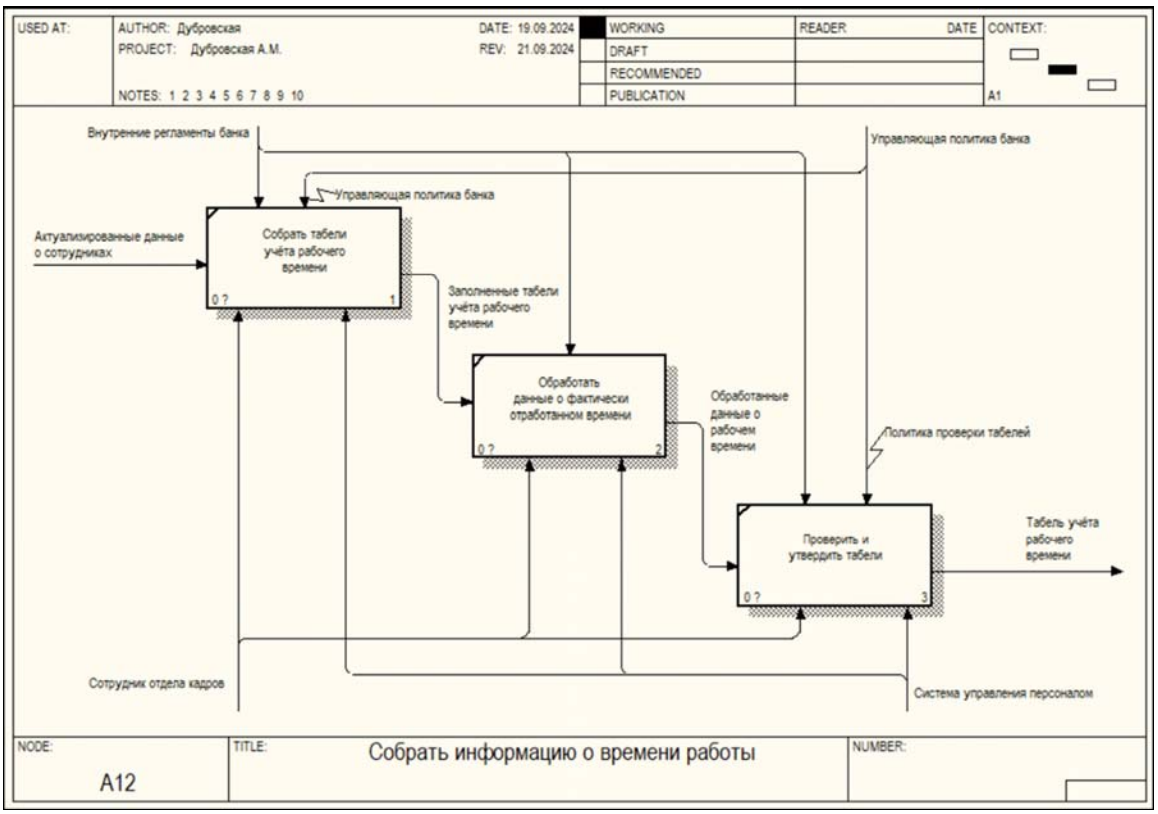


Рисунок 1.5 – Декомпозиция блока “Собрать информацию о времени работы”

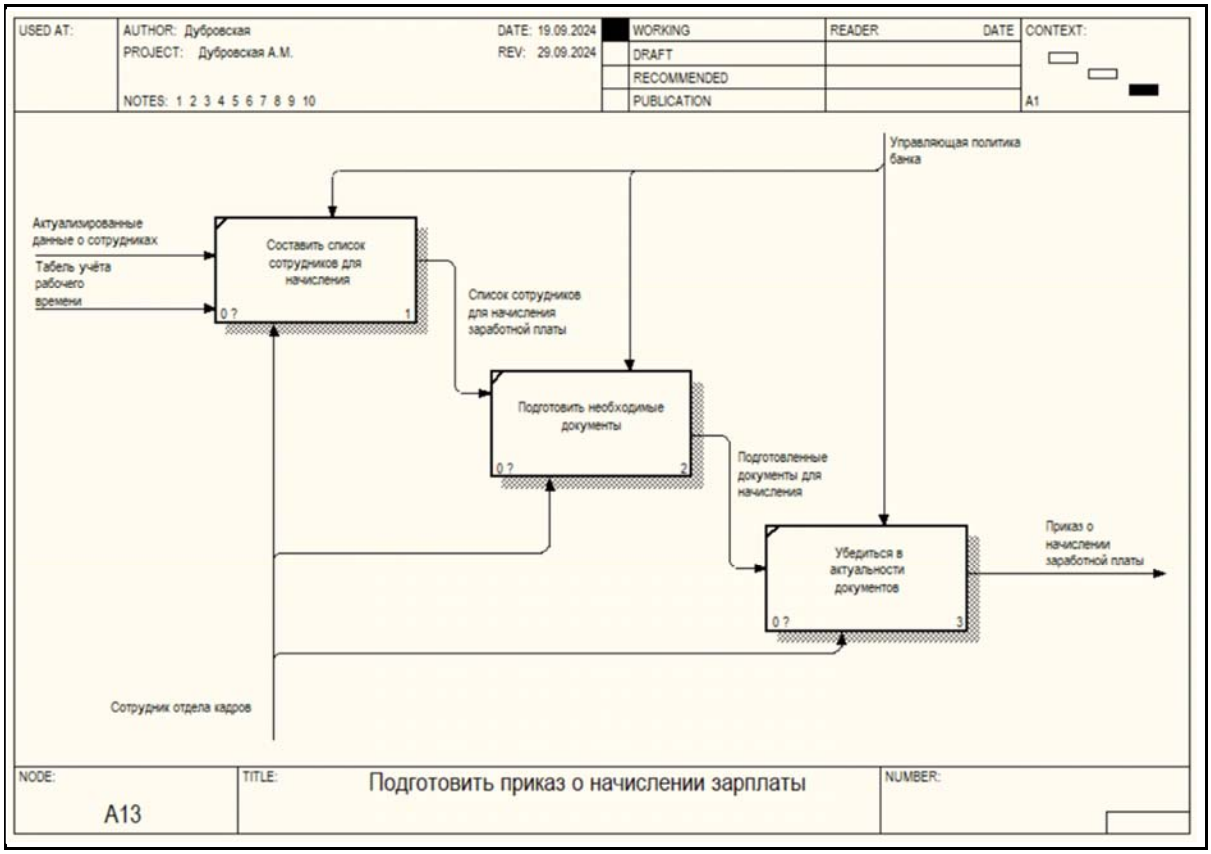


Рисунок 1.6 – Декомпозиция блока “Подготовить приказ о начислении зарплаты”

Рисунок 1.7 показывает декомпозицию блока “Рассчитать заработную плату”. Начальным является процесс “Определить сумму заработной платы”. Вторым является процесс “Включить в сумму дополнительные начисления и вычеты”. Последним процессом является процесс “Подготовить ведомость по расчёту заработной платы”. Для расчёта суммы зарплаты определяются оклады и рабочие часы, рассчитывается базовая зарплата, подсчитываются итоговые начисления. Данные процессы изображены на рисунке 1.8. Процессы последних двух блоков изображены на рисунках 1.9 и 1.10.

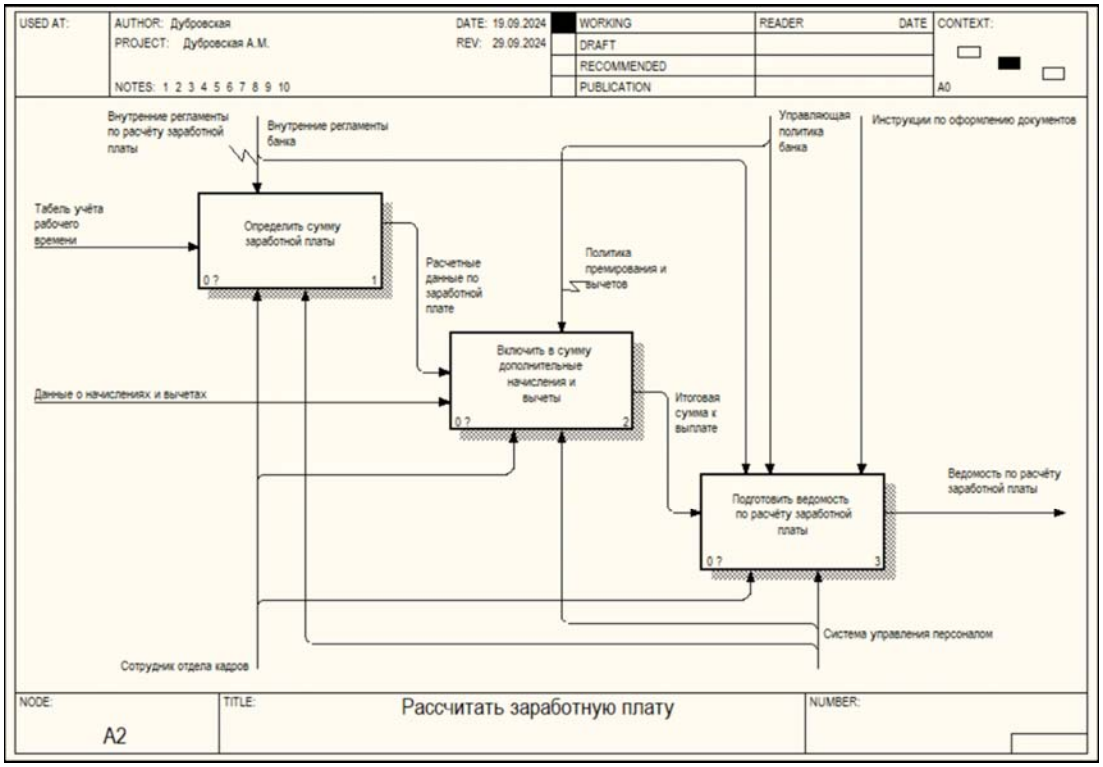


Рисунок 1.7 - Декомпозиция блока “Рассчитать заработную плату”

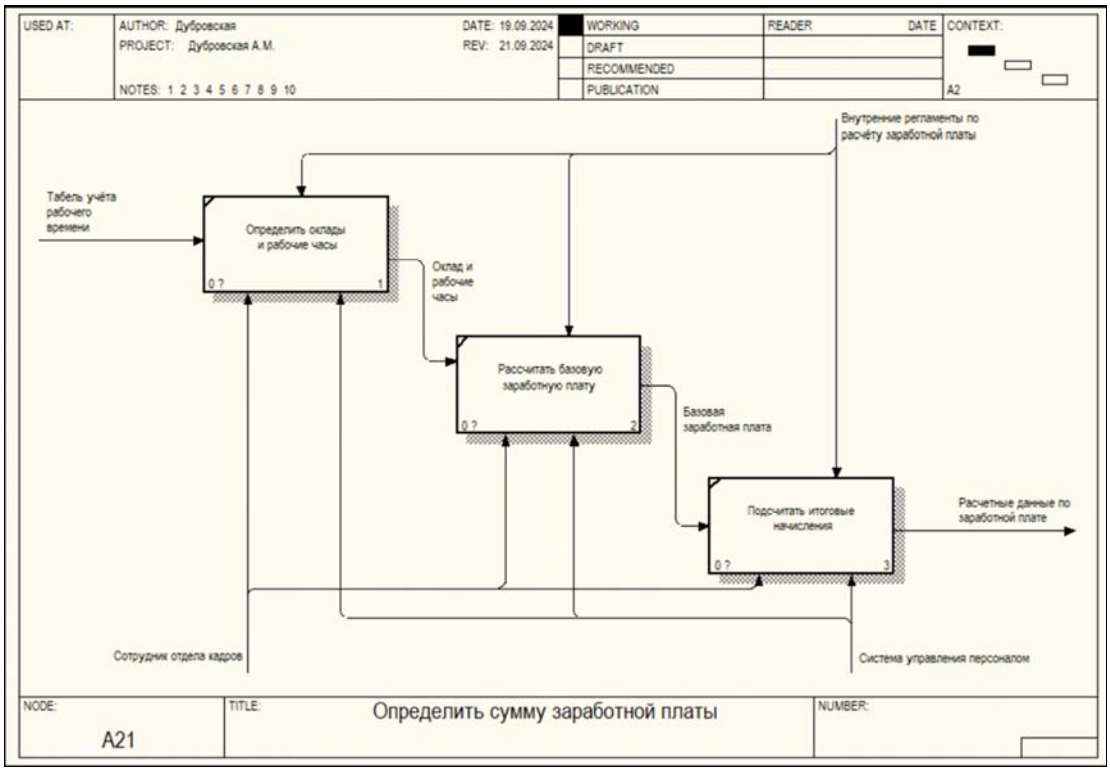


Рисунок 1.8 – Декомпозиция блока “Определить сумму заработной платы”

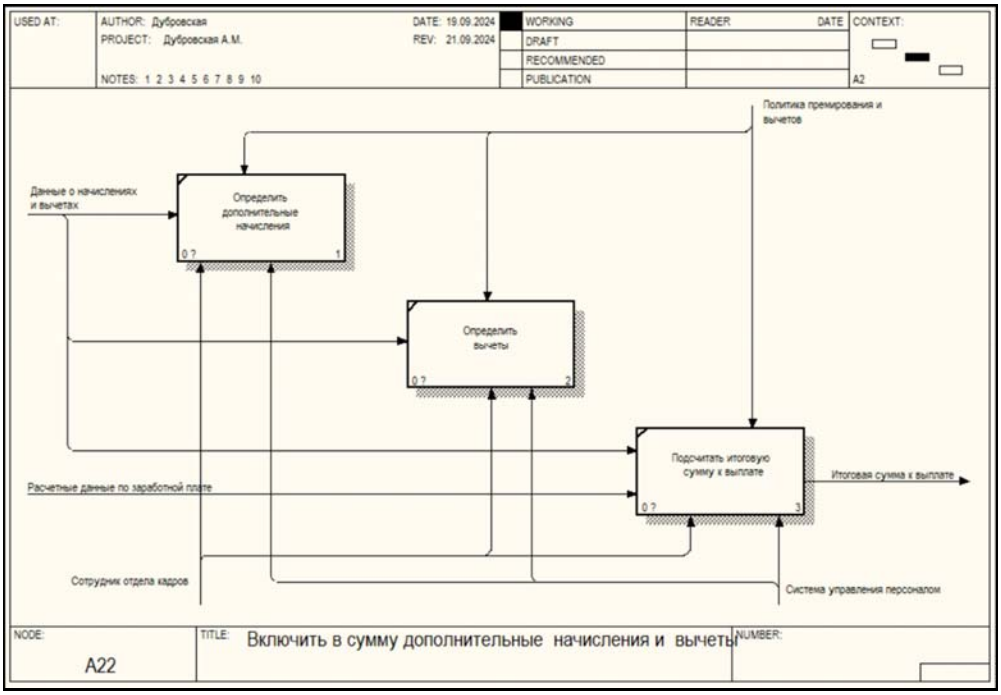


Рисунок 1.9 – Декомпозиция блока “Включить в сумму дополнительные начисления и вычеты”

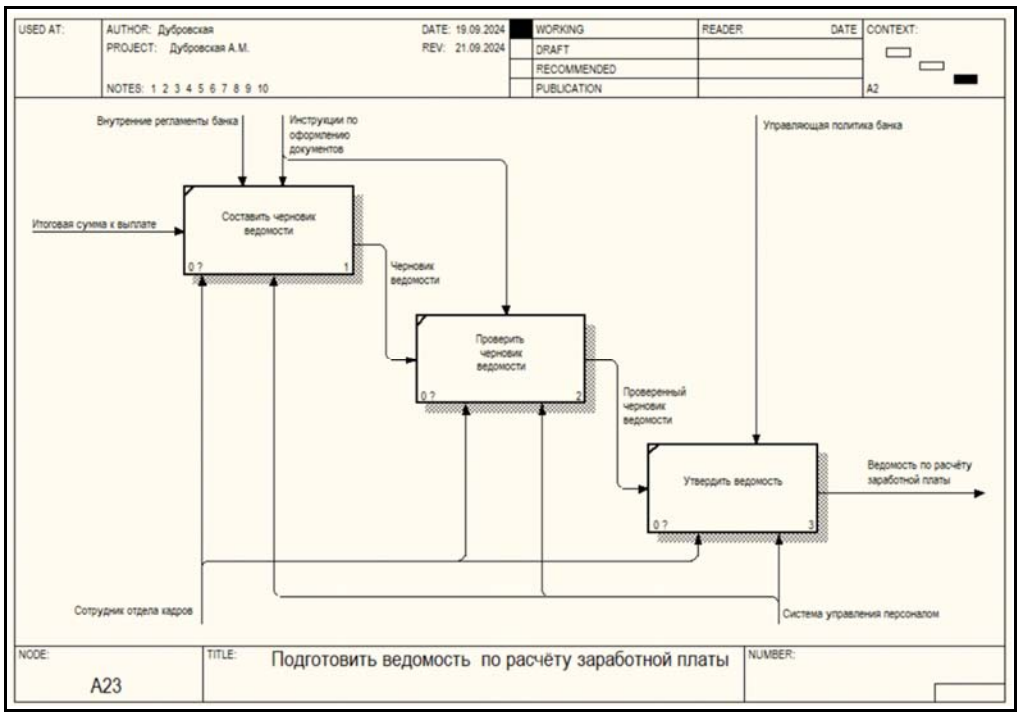


Рисунок 1.10 – Декомпозиция блока “Подготовить ведомость по расчёту заработной платы

Для исполнения процесса “Оформить начисление заработной платы”, который изображён на рисунке 1.11, необходимо выполнить следующие задачи:

– выполнить начисление заработной платы;

– сформировать отчёт по начисленной заработной плате;

– архивировать отчётность.

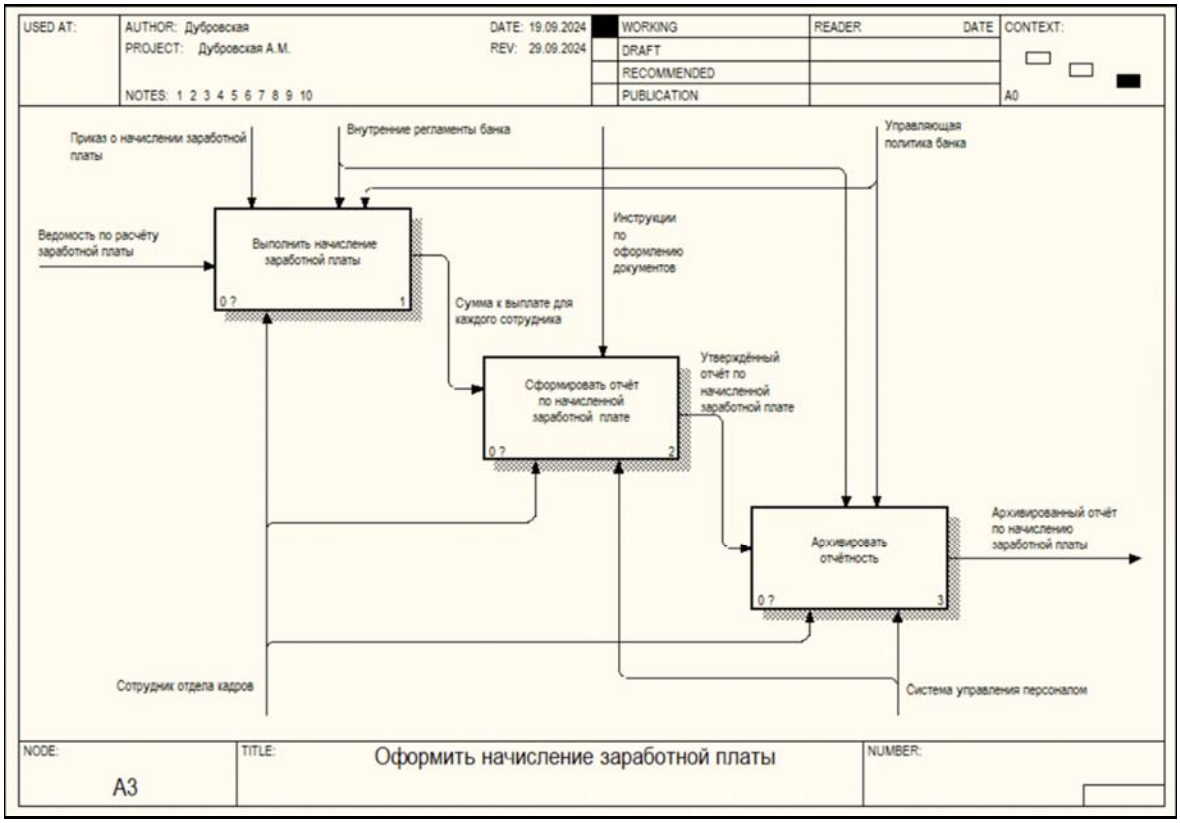


Рисунок 1.12 - Декомпозиция блока “Оформить начисление заработной платы”

1.3. Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Диаграмма вариантов использования является важным инструментом в процессе проектирования и разработки системы. Она помогает описать различные сценарии поведения системы на основе взаимодействия с внешними агентами, такими как пользователи или другие системы.

В данной работе представлена диаграмма вариантов использования для системы управления персоналом в банковской сфере, изображённая на рисунке 1.13. На этой диаграмме выделяются шесть основных актёров: пользователь, соискатель, сотрудник, сотрудник отдела кадров, сотрудник отдела безопасности, руководитель отдела и администратор.

Варианты использования для пользователя включают в себя возможность просмотра данных профиля, регистрации и авторизации. Эти функции обеспечивают базовый уровень взаимодействия с системой.

Сотрудник, наследует функции от пользователя и имеет возможность вести учёт рабочего времени, отобразить график учёта рабочего времени, изменять статус своих задач. Кроме того, сотрудник может подтвердить получение заработной платы.

Сотрудник отдела кадров, наследующий функции от сотрудника, выполняет разнообразные операции, такие как составление отчёта на выдачу заработной платы, фиксация данных о принятии и увольнении сотрудников, перевод сотрудников в другие отделы и отслеживание выдачи заработной платы. Он также имеет право изменять статус резюме соискателя и рассчитывать заработную плату. Работа с личными делами сотрудников включает в себя управление больничными, что включает просмотр информации о больничных и регистрацию больничных листов, а также управление отпусками, включая обработку заявок на отпуск.

Руководитель отдела, который наследует функции сотрудника, может оценивать качество работы сотрудников. Также руководитель может отслеживать время работы своих подчинённых, редактировать график работы отдела, назначать рабочие задачи для сотрудников, а также работать с документами сотрудников.

Сотрудник отдела безопасности может вносить записи в журнал инцидентов, а также проверять сотрудников на безопасность.

Соискатель, наследуя функции пользователя, может подавать резюме, записываться на собеседование и отслеживать статус своего резюме. Это обеспечивает активное участие соискателя в процессе трудоустройства.

Администратор системы наследуется от сотрудника и имеет свои уникальные варианты использования, включая работу с отделами и их расписаниями.

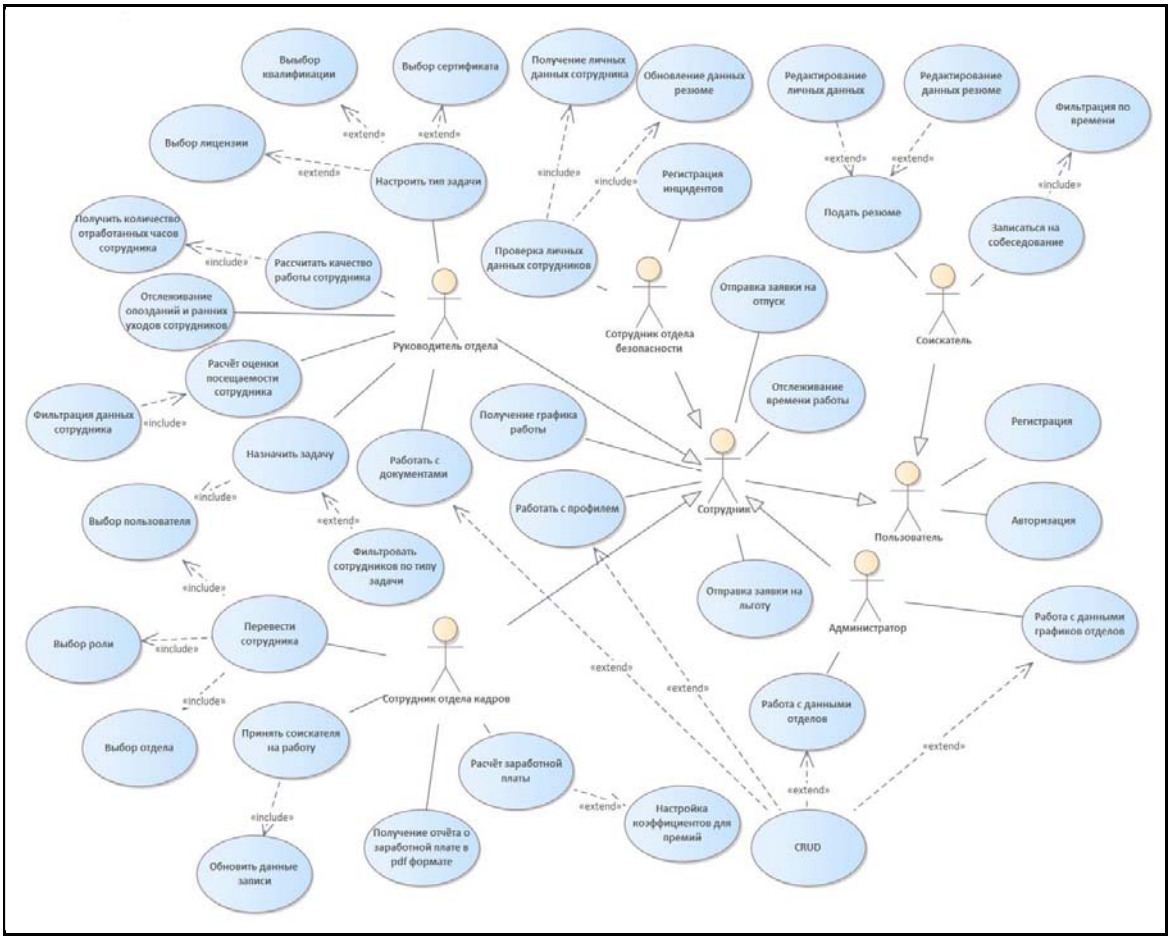


Рисунок 1.13 – Диаграмма вариантов использования

1.4. Разработка информационной модели предметной области

Информационная модель, изображённая на рисунке 1.14, содержит сводную информацию о структуре базы данных.

Сущность ЛИЧНЫЕ ДАННЫЕ (personal_data) связана со следующими сущностями:

– Сотрудники (employees) – идентифицирующая связь типа один-ко-одному, при этом каждому кортежу ЛИЧНЫЕ ДАННЫЕ должен соответствовать один кортеж Сотрудники.

– Квалификации (qualifications) – идентифицирующая связь типа один-ко-многим, при этом одному кортежу ЛИЧНЫЕ ДАННЫЕ может соответствовать несколько кортежей Квалификации;

– Инциденты (incidents) – идентифицирующая связь типа один-ко-многим, при этом одному кортежу ЛИЧНЫЕ ДАННЫЕ может соответствовать несколько кортежей Инциденты;

– Резюме (resumes) – идентифицирующая связь типа один-ко-одному, при этом каждому кортежу ЛИЧНЫЕ ДАННЫЕ может соответствовать один кортеж Резюме.

Атрибуты сущности ЛИЧНЫЕ ДАННЫЕ (personal_data):

– user_id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации каждого сотрудника. Значение автоматически увеличивается при добавлении нового сотрудника;

– full_name – строковое поле длиной до 150 символов, обязательное для заполнения. Хранит полное имя сотрудника;

– date_of_birth – поле типа DATE, обязательное для заполнения. Хранит дату рождения сотрудника;

– phone – строковое поле длиной до 15 символов. Хранит номер телефона сотрудника;

– email – строковое поле длиной до 100 символов, обязательное для заполнения. Хранит адрес электронной почты сотрудника;

– employee_data_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees.

Сущность СОТРУДНИКИ (employees) связана со следующими сущностями:

– ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа один-ко-одному;

– Отдел (department) – идентифицирующая связь типа многие-ко-одному, при этом одному кортежу СОТРУДНИКИ может соответствовать один кортеж Отдел;

– Рабочее время (work_time) – идентифицирующая связь типа один-ко-многим, при этом одному кортежу СОТРУДНИКИ может соответствовать несколько кортежей Рабочее время;

– Зарплата (payroll) – идентифицирующая связь типа один-ко-многим, при этом одному кортежу СОТРУДНИКИ может соответствовать несколько кортежей Зарплата;

– Инциденты (incidents) – идентифицирующая связь типа один-ко-многим.

Атрибуты сущности СОТРУДНИКИ (employees):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации каждого сотрудника. Значение автоматически увеличивается при добавлении нового сотрудника;

– hire_date – поле типа DATE, обязательное для заполнения. Хранит дату приема на работу;

– position – строковое поле длиной до 100 символов. Хранит должность сотрудника;

– termination_date – поле типа DATE. Хранит дату увольнения сотрудника;

– department_id – внешний ключ, ссылается на идентификатор отдела в таблице department;

Сущность ОТДЕЛ (представленная таблицей department) связана со следующими сущностями:

– СОТРУДНИКИ (employees) – идентифицирующая связь типа один-ко-многим;

– Рабочий график (work_schedule) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ОТДЕЛ (department):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации каждого департамента. Значение автоматически увеличивается при добавлении нового департамента;

– name – строковое поле длиной до 255 символов, обязательное для заполнения. Хранит название отдела;

– work_schedule_id – внешний ключ, ссылается на идентификатор рабочего графика в таблице work_schedule;

Сущность РАБОЧИЙ ГРАФИК (work_schedule) связана с сущностью:

ОТДЕЛ (department) – идентифицирующая связь типа один-ко-многим.

Атрибуты сущности РАБОЧИЙ ГРАФИК (work_schedule):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации рабочего графика. Значение автоматически увеличивается при добавлении нового графика.

– start_time – поле типа TIME, хранит время начала рабочего дня;

– end_time – поле типа TIME, хранит время окончания рабочего дня.

Сущность ЗАРПЛАТА (payroll) связана со следующими сущностями:

– СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ЗАРПЛАТА (payroll):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации записи о зарплате. Значение автоматически увеличивается при добавлении новой записи;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– base_salary – поле типа DECIMAL, хранит базовую зарплату сотрудника;

– total_salary – поле типа DECIMAL, хранит общую зарплату с учетом бонусов;

– pay_date – поле типа DATE, хранит дату выплаты зарплаты;

– bonuses – внешний ключ, ссылается на идентификатор бонусов в таблице bonus_deductions.

Сущность ИНЦИДЕНТЫ (incidents) связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ИНЦИДЕНТЫ (incidents):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации инцидента. Значение автоматически увеличивается при добавлении нового инцидента;

– description – текстовое поле, хранит описание инцидента;

– date – поле типа DATETIME, хранит дату и время инцидента;

– user_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– severity – перечисляемый тип, определяет степень серьезности инцидента. Возможные значения: 'LOW', 'MEDIUM', 'HIGH'.

Сущность РЕЗЮМЕ (resumes) связана с сущностью ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности РЕЗЮМЕ (resumes):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации резюме. Значение автоматически увеличивается при добавлении нового резюме;

– candidate_id – внешний ключ, ссылается на идентификатор кандидата в таблице personal_data;

– summary – текстовое поле, хранит краткое содержание резюме;

– skills – текстовое поле, хранит информацию о навыках кандидата;

– experience – текстовое поле, хранит информацию об опыте работы;

– education – текстовое поле, хранит информацию об образовании;

– languages – текстовое поле, хранит информацию о языках, которыми владеет кандидат.

Сущность КВАЛИФИКАЦИИ (qualifications) связана с сущностью ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности КВАЛИФИКАЦИИ (qualifications):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации квалификации. Значение автоматически увеличивается при добавлении новой квалификации;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице personal_data;

– degree – строковое поле длиной до 100 символов. Хранит информацию о степени;

– institution – строковое поле длиной до 100 символов. Хранит название учебного заведения;

– graduation_year – поле типа YEAR. Хранит год окончания учебного заведения.

Сущность ЛИЦЕНЗИИ (licenses) связана с сущностью ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ЛИЦЕНЗИИ (licenses):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации лицензии. Значение автоматически увеличивается при добавлении новой лицензии;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице personal_data;

– license_name – строковое поле длиной до 100 символов. Хранит название лицензии;

– issued_by – строковое поле длиной до 100 символов. Хранит информацию о том, кем выдана лицензия;

– issue_date – поле типа DATE, может быть пустым. Хранит дату выдачи лицензии.

Сущность БОЛЬНИЧНЫЕ ЛИСТЫ (sick_leaves) связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности БОЛЬНИЧНЫЕ ЛИСТЫ (sick_leaves):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации больничного листа. Значение автоматически увеличивается при добавлении нового больничного листа;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– start_date – поле типа DATE, хранит дату начала больничного;

– end_date – поле типа DATE, хранит дату окончания больничного;

– reason – строковое поле длиной до 255 символов. Хранит причину больничного.

Сущность ОТПУСКА vacations) связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ОТПУСКА (vacations):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации отпуска. Значение автоматически увеличивается при добавлении нового отпуска;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– start_date – поле типа DATE, хранит дату начала отпуска;

– end_date – поле типа DATE, хранит дату окончания отпуска;

– status – перечисляемый тип, определяет статус отпуска. Возможные значения: 'PENDING', 'APPROVED', 'REJECTED';

Сущность ЗАДАЧИ (tasks) связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности ЗАДАЧИ (tasks):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации задачи. Значение автоматически увеличивается при добавлении новой задачи;

– title – строковое поле длиной до 255 символов, обязательное для заполнения. Хранит название задачи;

– description – текстовое поле, хранит описание задачи;

– status – перечисляемый тип, определяет статус задачи. Возможные значения: 'Не начата', 'В процессе', 'Завершена';

– end_date – поле типа DATE, может быть пустым. Хранит дату завершения задачи;

– due_date – поле типа DATE. Хранит срок выполнения задачи;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees.

Сущность USERS (users) связана с сущностью ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа один-ко-одному.

Атрибуты сущности USERS (users):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации пользователя. Значение автоматически увеличивается при добавлении нового пользователя;

– username – строковое поле длиной до 50 символов, может быть пустым. Хранит имя пользователя;

– password – строковое поле длиной до 100 символов, может быть пустым. Хранит пароль пользователя;

– role – перечисляемый тип, определяет роль пользователя. Возможные значения: 'ADMIN', 'HR', 'DEPARTMENT_HEAD', 'CANDIDATE', 'EMPLOYEE';

– personal_data_id – внешний ключ, ссылается на идентификатор личных данных в таблице personal_data.

Сущность VACATIONS связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности VACATIONS (vacations):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации отпуска. Значение автоматически увеличивается при добавлении нового отпуска;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– start_date – поле типа DATE, хранит дату начала отпуска;

– end_date – поле типа DATE, хранит дату окончания отпуска;

– status – перечисляемый тип, определяет статус отпуска. Возможные значения: 'PENDING', 'APPROVED', 'REJECTED'.

Сущность WORK EXPERIENCE (work_experience) связана с сущностью ЛИЧНЫЕ ДАННЫЕ (personal_data) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности WORK EXPERIENCE (work_experience):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации опыта работы. Значение автоматически увеличивается при добавлении нового опыта;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице personal_data;

– company_name – строковое поле длиной до 100 символов, может быть пустым. Хранит название компании;

– position – строковое поле длиной до 100 символов. Хранит должность сотрудника в компании;

– start_date – поле типа DATE, хранит дату начала работы в компании;

– end_date – поле типа DATE, хранит дату окончания работы в компании.

Сущность WORK SCHEDULE (представленная таблицей work_schedule).

Атрибуты сущности WORK SCHEDULE (work_schedule):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации расписания. Значение автоматически увеличивается при добавлении нового расписания;

– start_time – поле типа TIME, хранит время начала рабочей смены;

– end_time – поле типа TIME, хранит время окончания рабочей смены;

Сущность WORK TIME (work_time) связана с сущностью СОТРУДНИКИ (employees) – идентифицирующая связь типа многие-ко-одному.

Атрибуты сущности WORK TIME (work_time):

– id – первичный ключ таблицы, представляет собой целое число типа INT и используется для уникальной идентификации записи о времени работы. Значение автоматически увеличивается при добавлении новой записи;

– employee_id – внешний ключ, ссылается на идентификатор сотрудника в таблице employees;

– start_time – поле типа TIME, хранит время начала рабочего дня;

– end_time – поле типа TIME, хранит время окончания рабочего дня;

– date – строковое поле длиной до 45 символов. Хранит дату работы в формате строки.

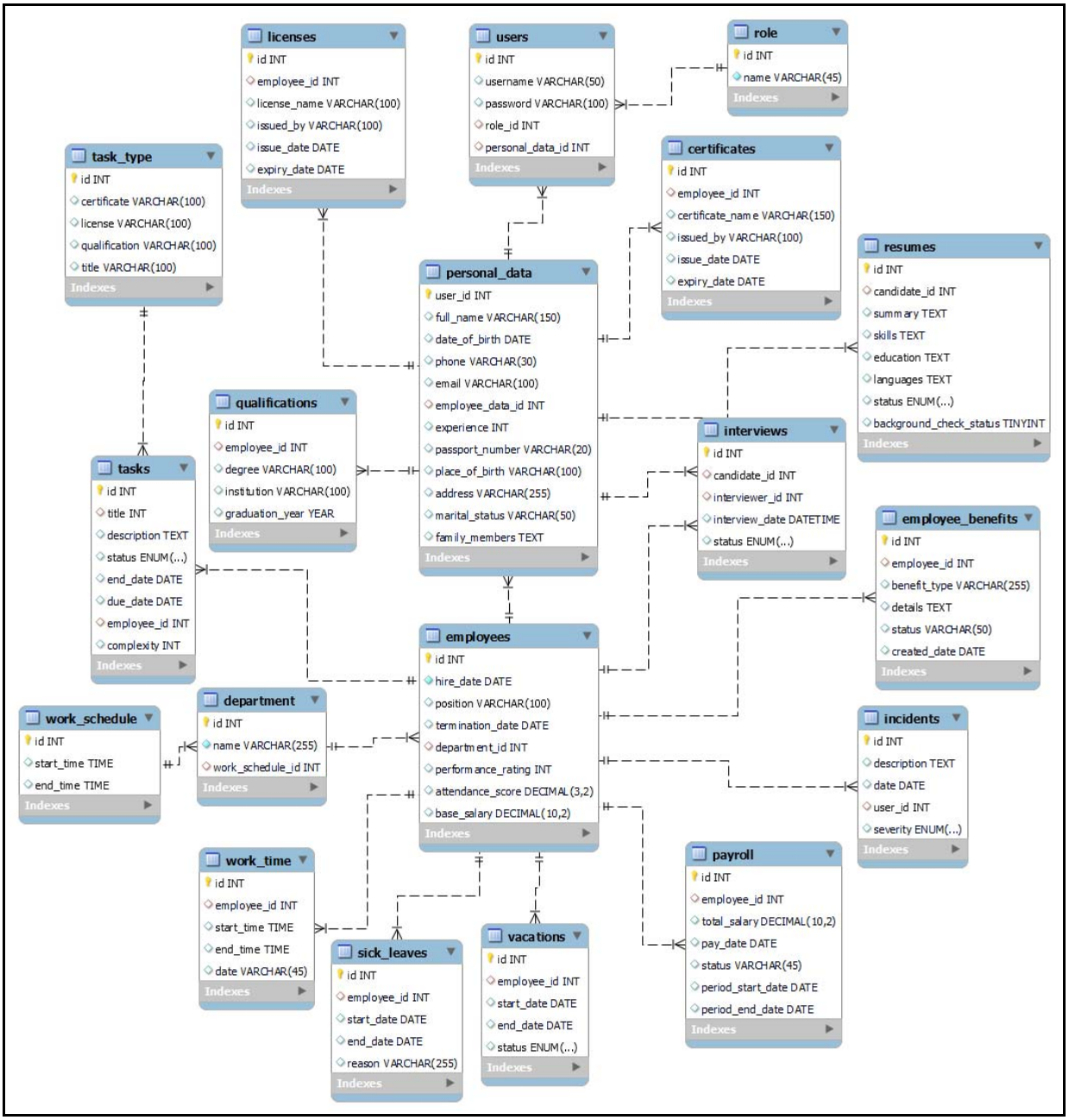


Рисунок 1.14 – Информационная модель

База данных соответствует третьей нормальной форме (3НФ) по следующим причинам.

Во-первых, каждая таблица имеет уникальный первичный ключ, что обеспечивает идентификацию каждой записи. Например, таблица employees использует поле id как первичный ключ, что гарантирует уникальность сотрудников. Это соответствует первой нормальной форме (1НФ), требующей, чтобы все атрибуты содержали атомарные значения.

Во-вторых, все неключевые атрибуты полностью функционально зависят от первичного ключа. Например, в таблице personal_data все поля, такие как full_name, date_of_birth и email, зависят только от user_id. Это исключает частичные зависимости, что является требованием второй нормальной формы (2НФ).

В-третьих, в таблицах отсутствуют транзитивные зависимости, которые могли бы нарушить условия 3НФ. Например, в таблице certificates все атрибуты, включая certificate_name и issued_by, зависят только от id. Нет зависимостей между неключевыми атрибутами, что также наблюдается в других таблицах, таких как tasks и vacations. В этих таблицах каждый неключевой атрибут, например, status или end_date, зависит только от своего первичного ключа и не влияет на другие поля.

Таким образом, структура базы данных, как видно из представленных таблиц, соответствует третьей нормальной форме. Это обеспечивает целостность данных, минимизирует избыточность и улучшает управляемость базы данных, что является ключевым при проектировании эффективных информационных систем.

1.5 UML-модели представления программного средства и их описание

1.5.1 Описание диаграммы последовательности

Диаграмма последовательности является мощным инструментом для визуализации взаимодействий между участниками системы в рамках конкретного сценария. Она иллюстрируют, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования [5].

Основные элементы диаграммы включают участников, представляющих собой объекты или роли в системе, линии жизни, отражающие жизненный цикл каждого объекта, и стрелки, иллюстрирующие пересылаемые сообщения между участниками. Анализируя диаграмму последовательности, можно глубже понять взаимодействие компонентов системы в контексте конкретного случая использования или прецедента.

На приведенной диаграмме последовательности, показанной на рисунке 1.15, отражается процесс расчета заработной платы. При открытии страницы с заработной платой сотруднику отдела кадров предлагается нажать кнопку "Рассчитать заработную плату". Эта операция инициирует вызов метода calculateSalary в классе SalaryCalculationController. Данный контроллер отправляет запрос на сервер, который обрабатывается в классе ClientThread через метод run, в котором реализованы все необходимые методы для работы с запросами.

Для обработки запроса на расчет заработной платы требуется добавить новую запись в базу данных. Для этого вызывается метод updateEntity в классе PayrollService, который отвечает за бизнес-логику работы с данными о заработной плате. Далее происходит вызов метода update в классе PayrollDAO, предназначенном для взаимодействия с сущностью "Заработная плата" в базе данных. Метод update создает запрос для добавления новой записи о заработной плате и возвращает созданный объект.

После этого объект заработной платы возвращается в ClientThread, где вызывается метод calculateSalary, выполняющий расчет заработной платы. После завершения расчета снова вызываются методы, аналогичные тем, что использовались для создания записи, но теперь запрос направлен на обновление данных о заработной плате в базе данных, что позволяет внести итоговую сумму в таблицу.

По завершении обновления объект возвращается в SalaryCalculationController, где вызывается метод viewSalaryData, после чего в интерфейсе отображается информация о рассчитанной заработной плате. Таким образом, данная диаграмма последовательности наглядно демонстрирует все ключевые шаги и взаимодействия, происходящие в процессе расчета заработной платы.

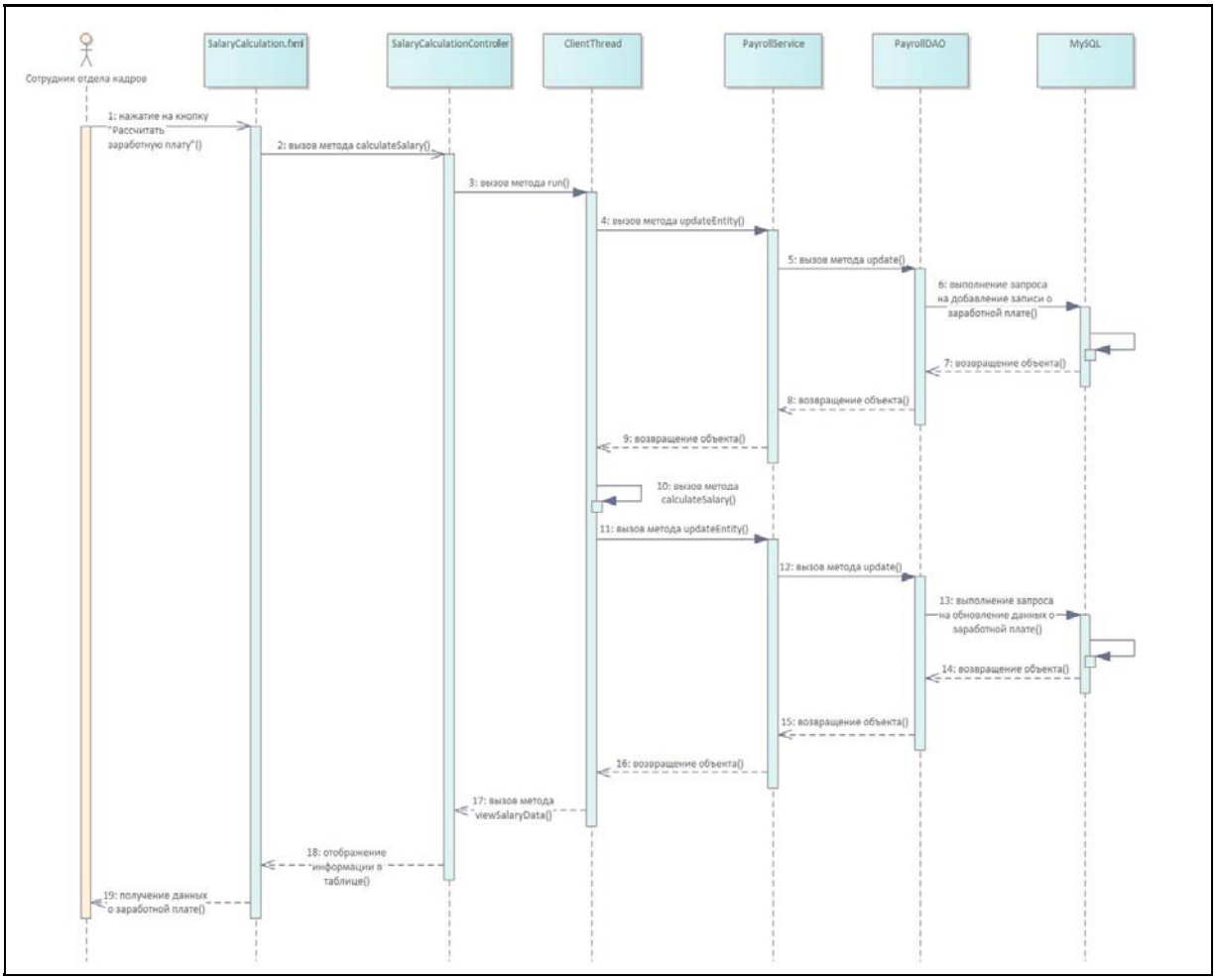


Рисунок 1.15 – Диаграмма последовательности расчёта заработной платы

1.5.2 Описание диаграммы деятельности

Далее будет описана диаграмма деятельности. По сути, диаграмма деятельности представляет собой блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой. В отличие от традиционной блок-схемы диаграмма деятельности показывает параллелизм так же хорошо, как и ветвление потока управления [6]. Диаграмма деятельности, которая изображена на рисунке 1.16, отображает последовательность действий для оценки качества работы сотрудников в системе управления персоналом, где все действия выполняются от имени руководителя отдела. Процесс начинается с выбора конкретного сотрудника и получения данных о его работе. Затем руководитель анализирует данные о времени работы, инцидентах и выполненных задачах. После этого вычисляется итоговая оценка по 10-бальной шкале, и сохранение оценки и комментариев завершается процесс.

Рисунок 1.16 – Диаграмма деятельности

1.5.3 Описание диаграммы состояния

Диаграмма состояний показывает порядок состояний и событий, возможный в рамках данной системы для одного класса объектов [7]. На рисунке 1.17 представлена диаграмма состояния “Обработка резюме”. Данная диаграмма иллюстрирует основные действия, выполняемые сотрудником отдела кадров в процессе обработки резюме кандидатов.

Обработка резюме начинается, когда сотрудник отдела кадров получает новое резюме. После этого он переходит к состоянию “Просмотр резюме”, где анализирует информацию кандидата. Затем сотрудник оценивает соответствие кандидата требованиям вакансии, что приводит его к состоянию “Оценка соответствия требованиям”.

Если резюме соответствует требованиям, сотрудник переходит к оценке личных качеств кандидата в состоянии “Оценка качеств”. В зависимости от результатов этих оценок, принимается решение: резюме может быть одобрено или отклонено. Если резюме одобрено, сотрудник фиксирует это решение и переходит к состоянию “Запись на собеседование”, где заполняет информацию о времени и дате собеседования.

Если же резюме отклонено, сотрудник фиксирует причины отклонения и уведомляет кандидата, после чего процесс завершается. Если в процессе обработки резюме возникают ошибки, например, резюме не соответствует требованиям или качествам, сотрудник возвращается к предыдущим состояниям для повторной оценки или информирования кандидата.

Таким образом, диаграмма состояния “Обработка резюме” четко описывает все этапы и действия, происходящие в процессе анализа и принятия решения по резюме кандидатов.

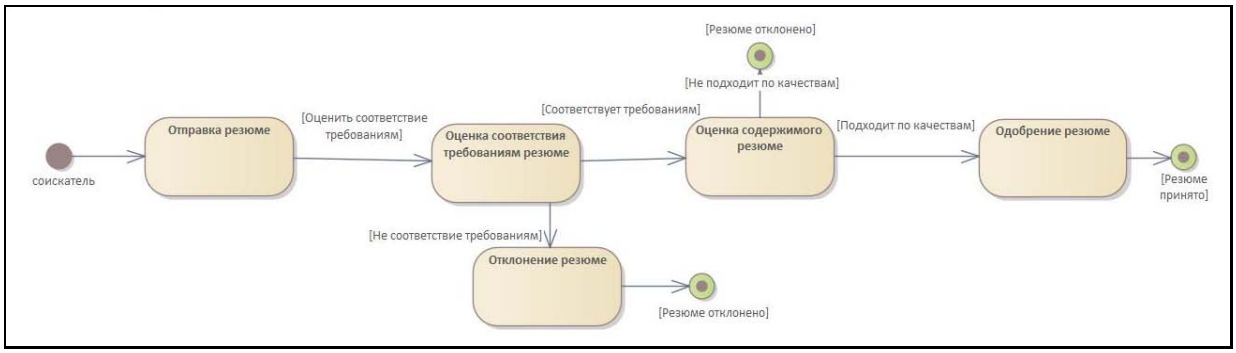


Рисунок 1.17 – Диаграмма состояния “Обработка резюме”

1.5.4 Описание диаграммы классов

Далее будут описаны диаграммы классов, использованных в системе. Диаграммы классов детально представляют различные виды взаимодействия отдельных классов системы [8]. Диаграмма классов, изображённая на рисунке 1.18, представляет основные сущности системы для связи с бд и передачи данных объектов между клиентом и сервером.

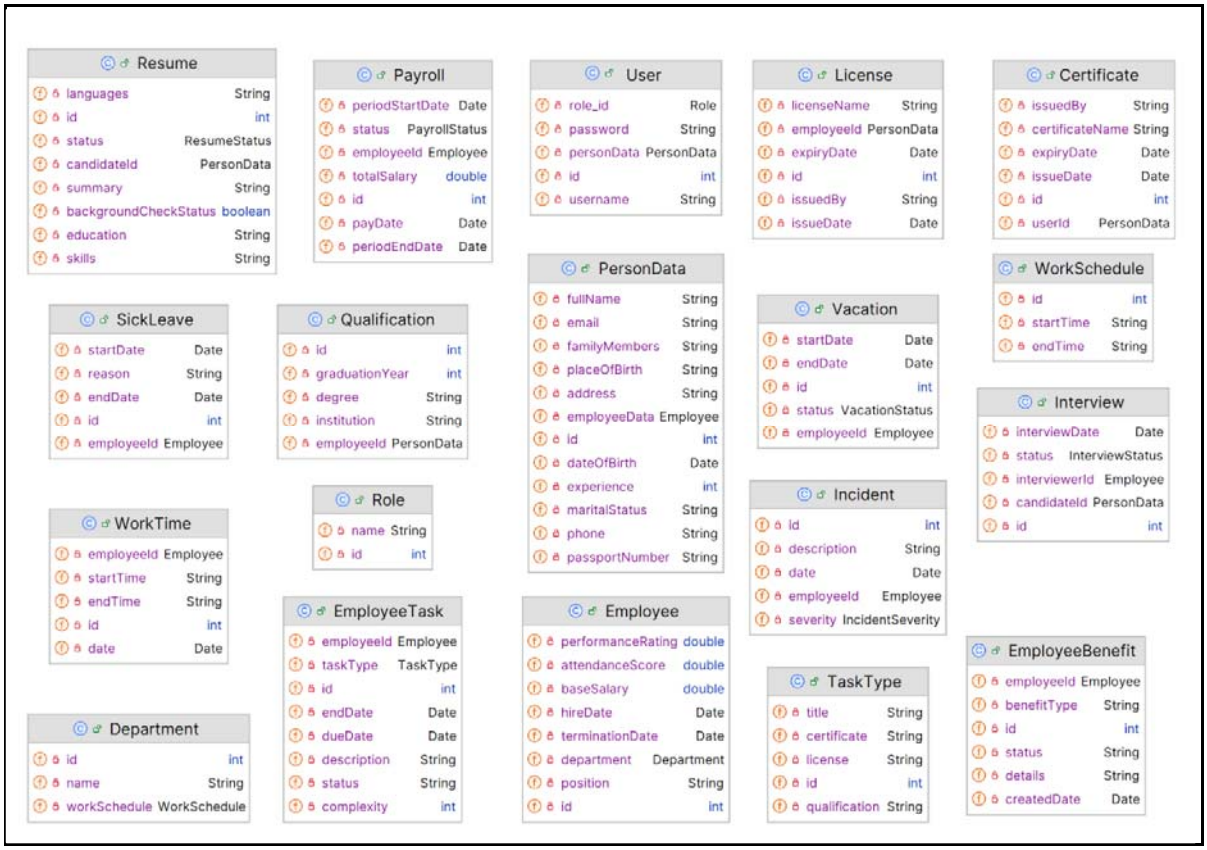


Рисунок 1.18 – Диаграмма классов

Диаграммы классов контроллеров общих страниц представлены на рисунке 1.19.

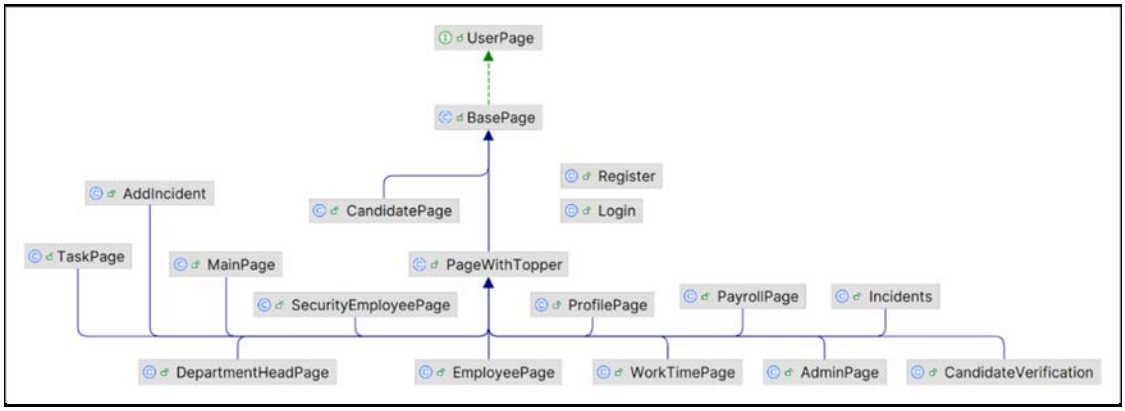


Рисунок 1.19 – Диаграмма классов общих контроллеров

На рисунке 1.20 представлены контроллеры руководителя отдела, а на рисунке 1.21 – сотрудника отдела кадров.

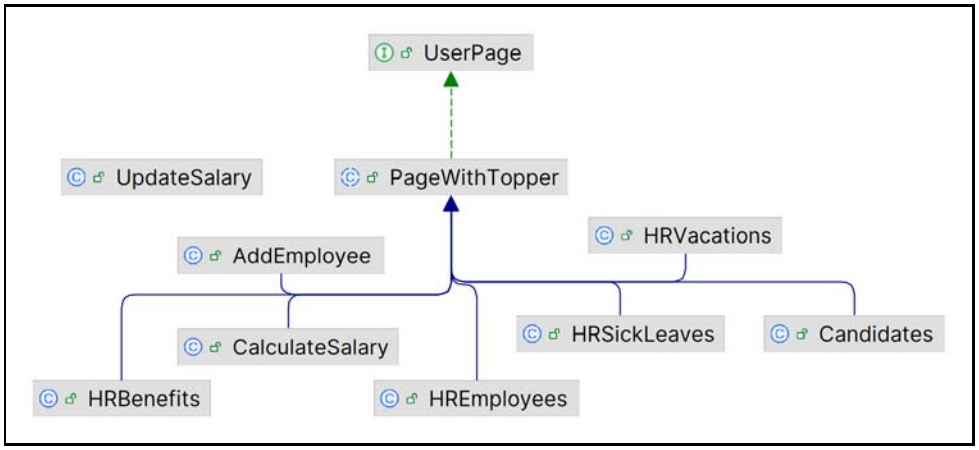


Рисунок 1.20 – Диаграмма классов контроллеров сотрудника отдела кадров

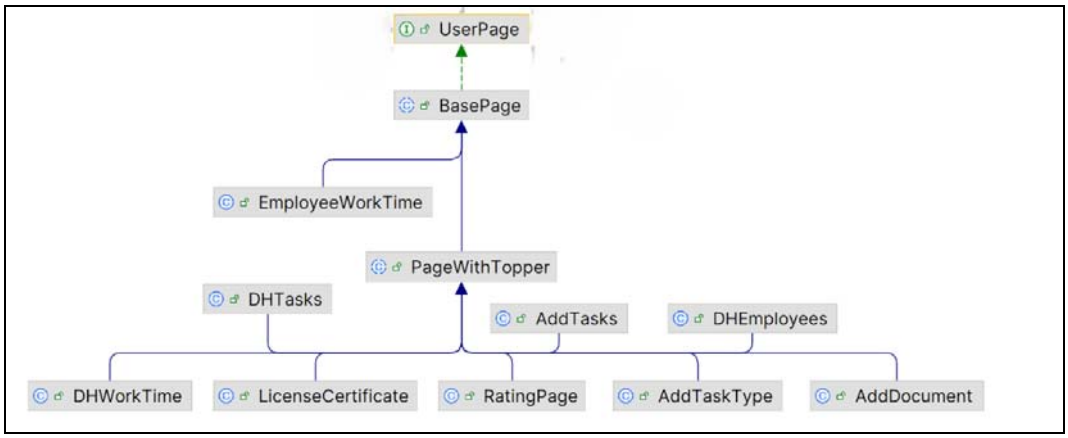


Рисунок 1.21 – Диаграмма классов контроллеров руководителя отдела

На рисунке 1.22 представлена диаграмма классов, реализующих соединение между клиентом и сервером, данные классы находятся в пакете TCP.

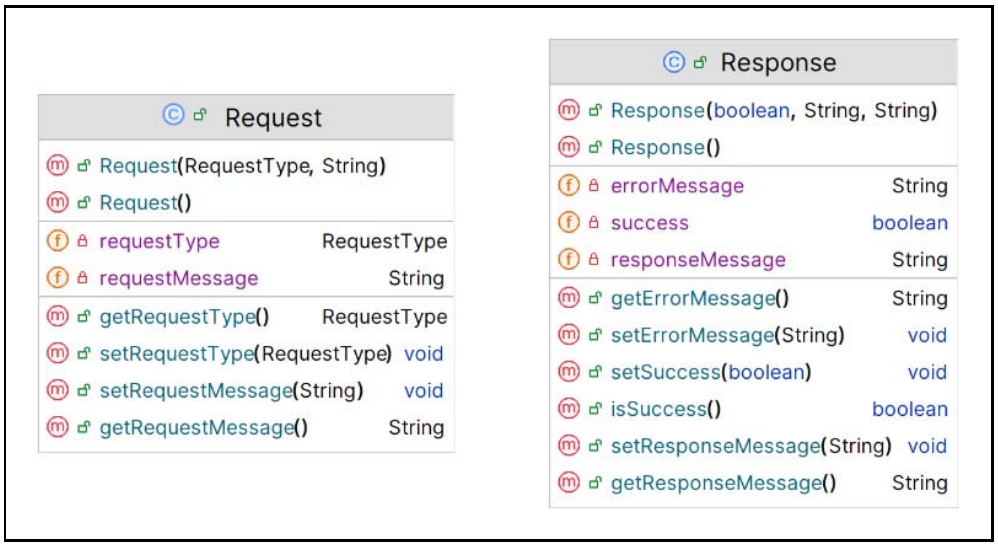


Рисунок 1.22 – Диаграмма классов пакета TCP

1.5.5 Описание диаграммы развёртывания

Далее будет описана диаграмма развёртывания системы. Такая диаграмма показывает конфигурацию узлов, где производится обработка информации, и то, какие компоненты размещены на каждом узле [9]. На диаграмме развертывания, показанной на рисунке 1.23, представлено взаимодействие трёх узлов: User PC, Server PC и Server DB. User PC включает в себя компонент Client.jar, который отображает приложение пользователю и устанавливает соединение с Server PC через протокол TCP/IP. Server PC работает в среде выполнения JVM и использует компоненты Server.jar, mysql-connector-java.jar (для подключения к MySQL), logback-classic.jar (для логирования), hibernate-core.jar (для работы с Hibernate), jackson-databind.jar (для работы с JSON), itext7-core.jar (для работы с PDF) и javafx-controls.jar (для пользовательского интерфейса JavaFX). Server PC также взаимодействует с Server DB, который представляет собой базу данных (например, MySQL), через JDBC для выполнения операций с данными.

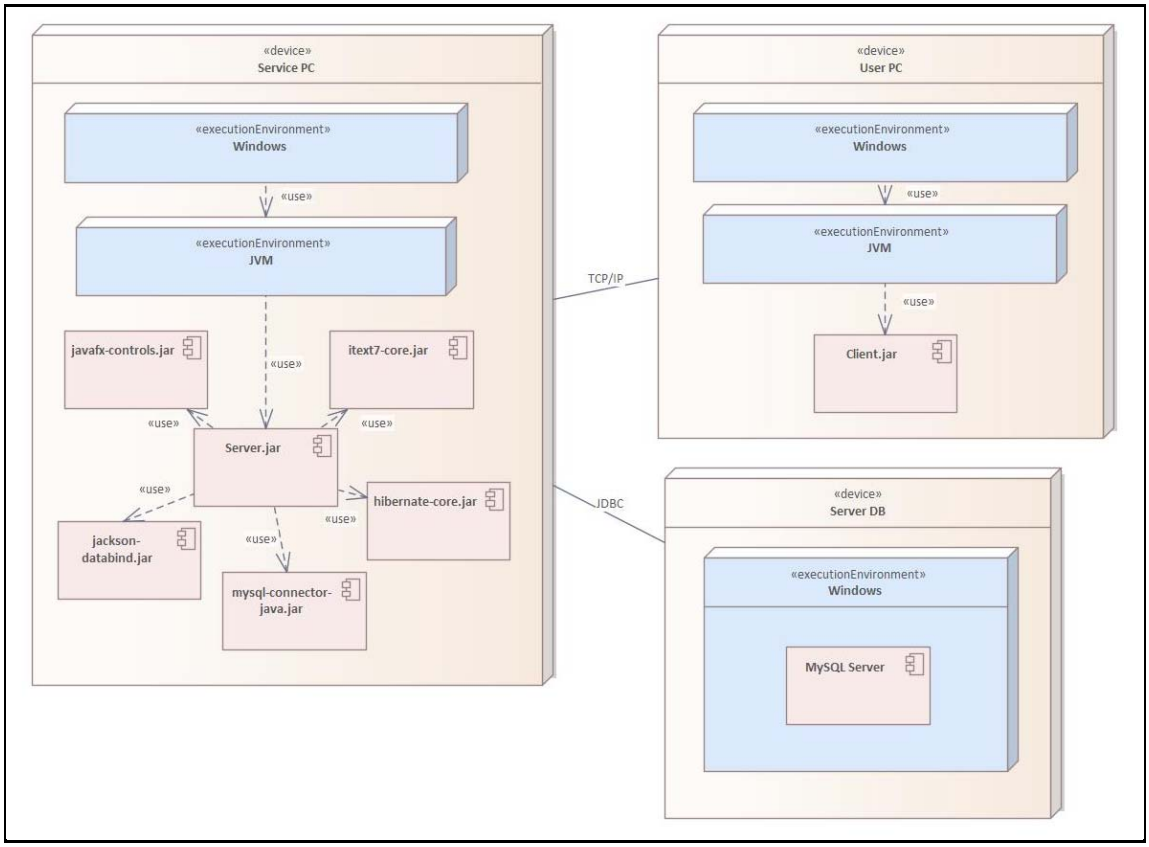


Рисунок 1.23 – Диаграмма развёртывания

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Постановка задачи

Для разработки системы управления персоналом в банковской сфере необходимо учитывать уникальные потребности и особенности данной области. Эффективное управление персоналом играет ключевую роль в обеспечении высоких результатов работы банка, так как человеческий ресурс является одним из самых важных активов. Система должна автоматизировать процессы учёта рабочего времени, расчёта заработной платы, а также управления кадровыми данными, что позволит существенно повысить эффективность работы HR-отдела.

Важным аспектом является возможность взаимодействия сотрудников с системой, что включает в себя функции просмотра личных данных, регистрации и авторизации. Система должна предоставлять сотрудникам возможность вести учёт рабочего времени, управлять статусами задач, а также подтверждать получение заработной платы. Это создаст основу для прозрачности и открытости в отношениях между работниками и руководством.

Сотрудники отдела кадров будут иметь доступ к расширенным функциям, включая составление ведомостей на выдачу заработной платы и обработку заявок на отпуск. Они также смогут отслеживать изменения в кадровом составе, что важно для поддержания актуальности информации и соблюдения законодательства.

Руководители отделов смогут оценивать производительность своих подчинённых, управлять графиками работы, назначать задачи и контролировать выполнение поставленных целей. Это даст возможность не только повышать эффективность работы команды, но и оперативно реагировать на возникающие проблемы.

Система также должна учитывать интересы соискателей, предоставляя им возможность подавать резюме и записываться на собеседования. Это создаст активное участие кандидатов в процессе трудоустройства и повысит шансы банка на привлечение квалифицированных специалистов.

Таким образом, разработка системы управления персоналом в банковской сфере позволит оптимизировать процессы, повысить уровень удовлетворенности сотрудников и обеспечить соблюдение всех необходимых стандартов и требований. Система будет использоваться на платформе, обеспечивающей надежный и безопасный доступ к данным, что особенно важно в условиях работы с финансовыми ресурсами.

2.2 Архитектурные решения

Разрабатываемая система управления персоналом в банковской сфере построена на языке программирования Java, который предоставляет мощные возможности для создания надежных, масштабируемых и кроссплатформенных приложений. Java является одним из самых популярных языков программирования благодаря своей стабильности и поддержке объектно-ориентированного программирования, что позволяет легко управлять сложными системами и обеспечивать их расширяемость.

Для создания пользовательского интерфейса использовалась библиотека JavaFX, которая предлагает современные инструменты для разработки графических интерфейсов. JavaFX позволяет создавать визуально привлекательные и интерактивные приложения с богатым функционалом. Использование JavaFX обеспечивает плавный пользовательский опыт и возможность интеграции мультимедийных элементов, что особенно важно для повышения удобства работы с системой.

Система использует MySQL в качестве базы данных, что гарантирует высокую производительность и надежность хранения данных. MySQL является одним из самых распространенных решений для управления базами данных, предлагая мощные функции для работы с большими объемами информации. Для взаимодействия с базой данных применяется Hibernate, который служит для упрощения процесса объектно-реляционного отображения (ORM). Это позволяет разработчикам работать с объектами Java вместо написания сложных SQL-запросов, что значительно ускоряет процесс разработки и уменьшает вероятность ошибок.

Подключение к базе данных осуществляется через протокол TCP/IP с использованием сокетов, что обеспечивает надежную и безопасную передачу данных между клиентом и сервером. Серверная часть приложения принимает входящие соединения и обрабатывает запросы клиентов в многопоточном режиме, что повышает отзывчивость системы и позволяет одновременно обслуживать множество пользователей. Использование сокетов позволяет поддерживать постоянное соединение, что критично для приложений, требующих быстрой обработки данных и минимальной задержки.

Кроме того, архитектура системы включает в себя использование шаблона проектирования DAO (Data Access Object), который отделяет бизнес-логику от логики доступа к данным. Это делает код более структурированным и облегчает его тестирование и сопровождение. DAO обеспечивает единый интерфейс для работы с данными, что позволяет легко изменять реализацию доступа к данным без воздействия на остальную часть приложения. Диаграмма классов данного шаблона изображена на рисунке 2.1.

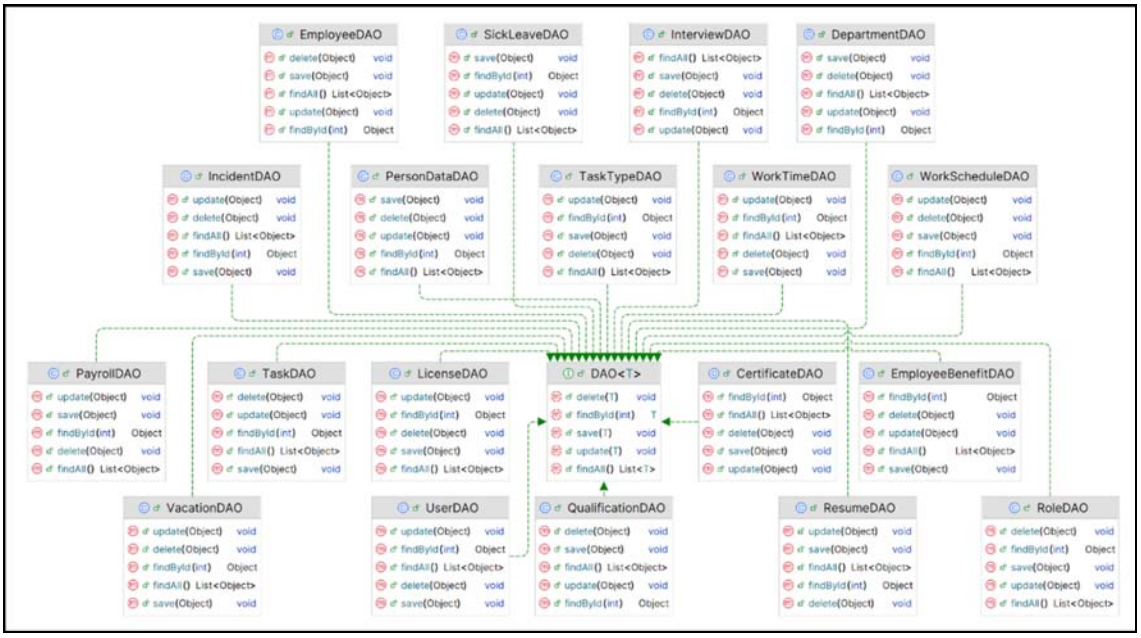


Рисунок 2.1 – Диаграмма классов шаблона проектирования DAO

Система управления персоналом также включает механизмы для обработки запросов и ответов, что позволяет клиентским приложениям взаимодействовать с сервером. Все необходимые настройки, такие как параметры подключения к базе данных, хранятся в конфигурационном файле, что делает их доступными для изменения пользователем. Это позволяет обеспечить гибкость в настройках и управлении системой.

Паттерн Template Method определяет общий алгоритм выполнения операций, оставляя некоторые шаги для реализации подклассам. В нашей системе этот паттерн реализован через интерфейс UserPage и абстрактный класс BasePage. Интерфейс задаёт методы, которые должны быть реализованы для работы с пользовательскими страницами, в то время как BasePage предоставляет общие функции, такие как загрузка пользовательского интерфейса и обновление информации о пользователе. Это позволяет подклассам, наследующим от BasePage, реализовывать специфические функции для каждой роли, сохраняя при этом общую структуру. Например, страницы для сотрудников и администраторов могут иметь уникальные элементы интерфейса, но использовать общие методы для навигации и обновления данных. Паттерн Template Method обеспечивает гибкость и расширяемость системы, позволяя легко добавлять новые страницы и функции без изменения существующей логики. Диаграммы, реализующие данный паттерн, изображены на рисунках 1.19, 1.20 и 1.21.

Кроме того, для реализации программы использовался паттерн команда. Для этого паттерна диаграмма классов изображена на рисунке 2.2.

Рисунок 2.2 – Диаграмма классов паттерна “Команда”

В программе также использовался паттерн “Шаблонный метод”. Шаблонный метод определяет основу алгоритма и позволяет переопределить некоторые шаги алгоритма, не изменяя его структуру в целом [10]. Диаграмма классов которого изображена на рисунке 2.3.

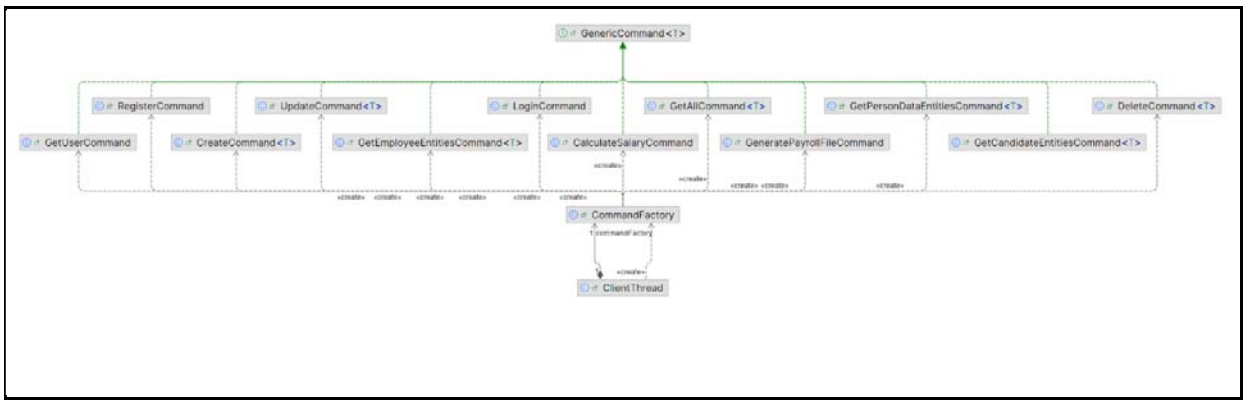


Рисунок 2.3 – Диаграмма классов паттерна “Шаблонный метод”

Таким образом, архитектурные решения, примененные в данной системе, создают основу для эффективного управления персоналом в банковской сфере, обеспечивая надежность, безопасность и высокую производительность.

2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства

На рисунках 2.4 – 2.7 изображена схема алгоритма работы программы. При запуске программы пользователю предлагается авторизоваться или зарегистрироваться если он не зарегистрирован. После успешного входа в систему пользователю в зависимости от роли открывается ряд возможностей. Если пользователь заходит от имени администратора, он может работать с данными сотрудников, отделами и их расписанием. Если же был выполнен вход от имени сотрудника, пользователь может работать с задачами, заработными платами и со своим профилем. Если авторизация была выполнена в качестве сотрудника отдела кадров, пользователю предлагается работать с заработными платами, данными сотрудников и соискателей, больничными листами, отпусками и льготами. Если выполнен вход под сотрудником отдела безопасности, ему предлагается работать с личными данными пользователей и инцидентами сотрудников. При входе от имени руководителя отдела, пользователю открываются возможности работы с задачами и их типами, данными сотрудников и данными о времени работы, а также возможность работы с документами сотрудников.

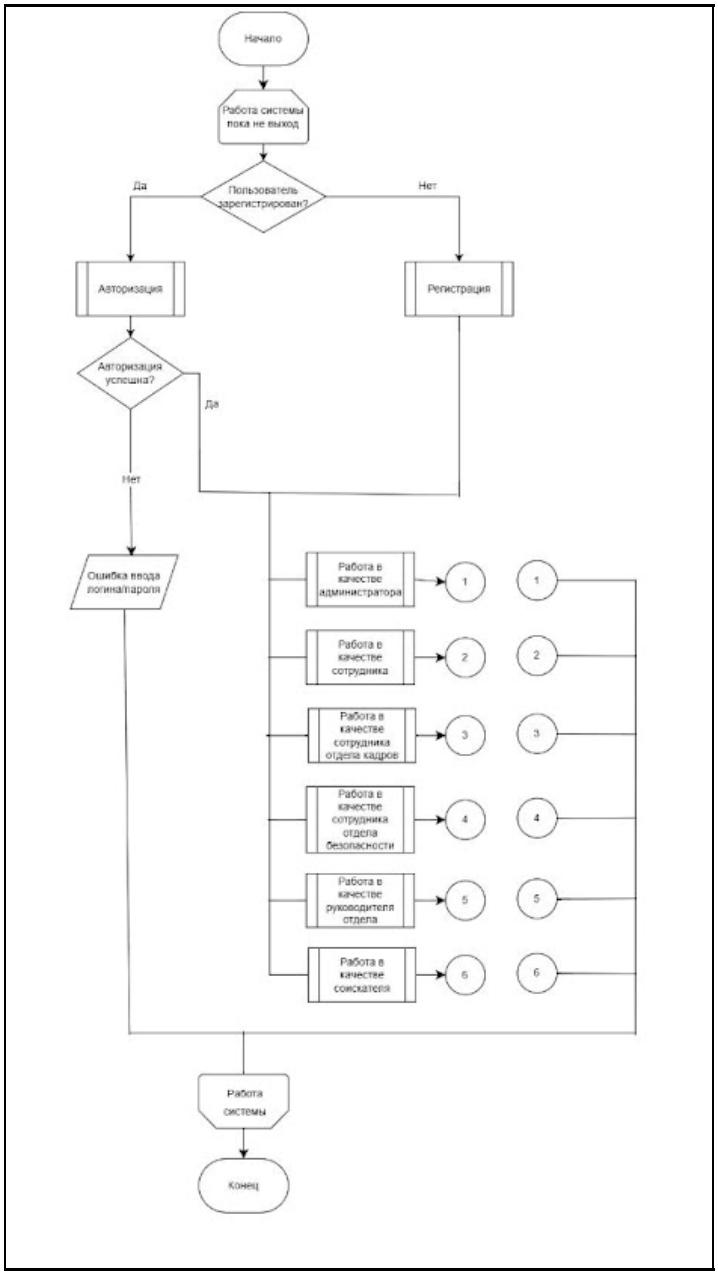


Рисунок 2.4 – Схема алгоритма работы программы, часть 1

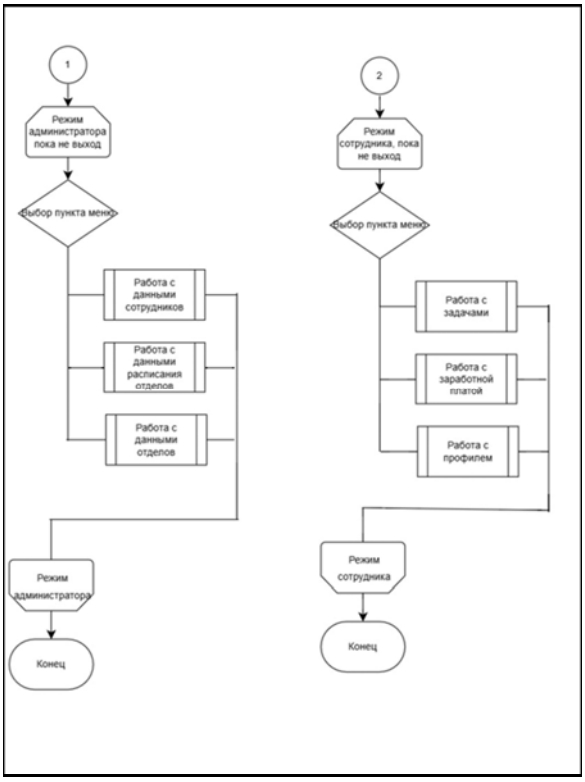


Рисунок 2.5 – Продолжение схемы алгоритма работы программы, часть 2

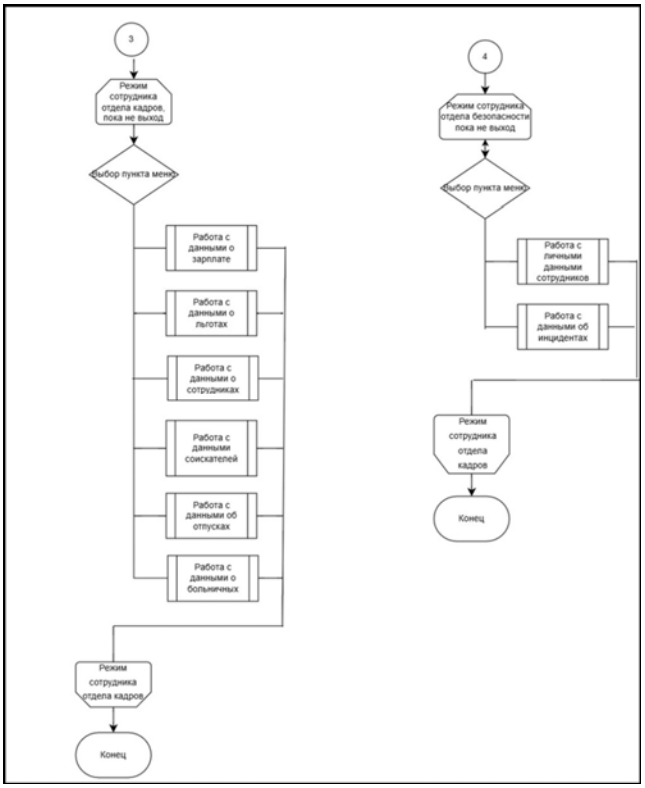


Рисунок 2.6 – Продолжение схемы алгоритма работы программы, часть 3

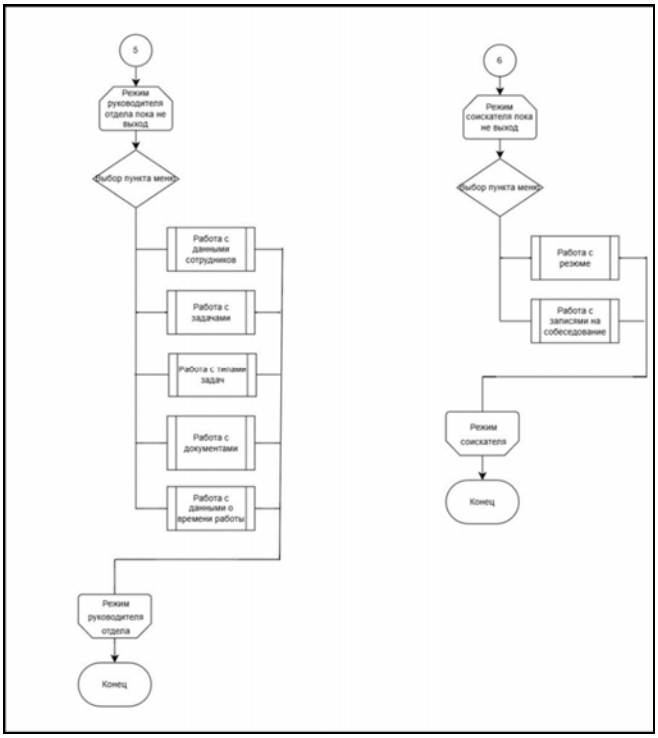


Рисунок 2.7 – Продолжение схемы алгоритма работы программы, часть 4

На рисунке 2.8 изображён процесс изменения статуса задачи сотрудника. При изменении статуса задачи в системе происходит последовательный процесс, который начинается с взаимодействия пользователя с интерфейсом и завершается обновлением данных на сервере.

Сначала пользователь нажимает на ячейку статуса задачи в таблице, что открывает выпадающий список с возможными статусами ("Не начата", "В процессе", "Завершена"). После выбора нового статуса, пользователь подтверждает изменения нажатием кнопки "Подтвердить". В этот момент система собирает информацию о всех задачах, включая их новые статусы.

Сформированный запрос для обновления статусов задач отправляется на сервер. Сервер, получив запрос, проверяет наличие задач для указанного работника. Если задачи найдены, сервер обновляет их в базе данных и формирует успешный ответ. В противном случае, сервер отправляет сообщение об отсутствии задач. Клиент получает ответ и обновляет интерфейс, отображая актуальные данные.

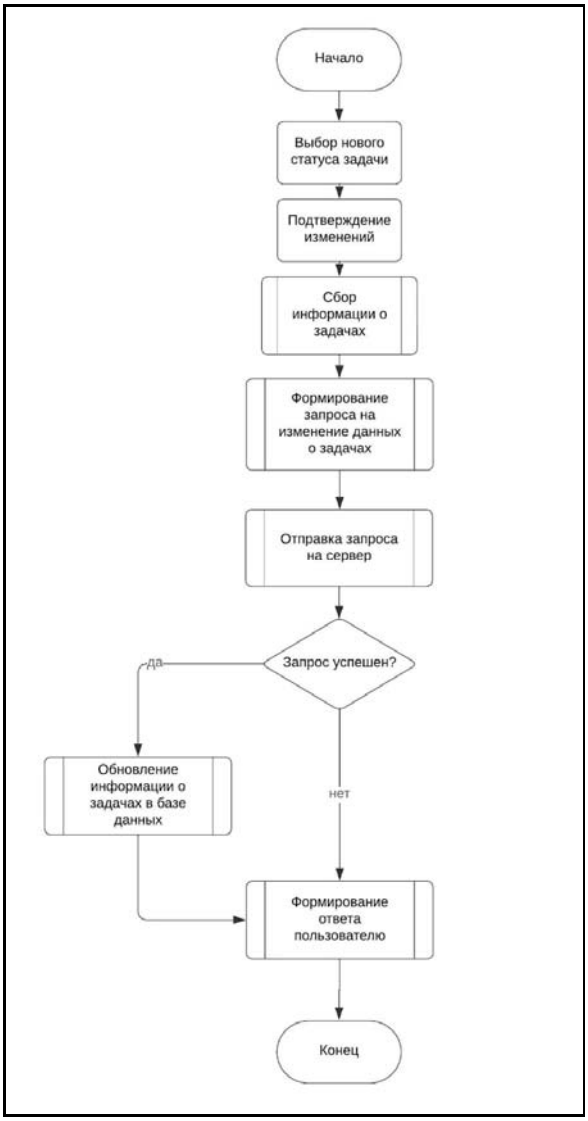


Рисунок 2.8 - Схема алгоритма изменения статуса задачи

На рисунке 2.9 показана блок-схема процесса авторизации. Процесс авторизации начинается с ввода пользователем имени пользователя и пароля в соответствующие поля интерфейса. При нажатии на кнопку "Войти" создается объект User, в который записываются введенные данные. Затем формируется запрос на авторизацию, который отправляется на сервер. Сервер получает запрос и извлекает данные пользователя, производя поиск в базе данных.

Если пользователь найден, сервер проверяет, совпадает ли введенный пароль с сохраненным в базе данных. В случае совпадения сервер формирует успешный ответ, включая данные пользователя, и отправляет его обратно клиенту. Если пароль не совпадает или пользователь не найден, сервер формирует ответ с сообщением об ошибке, указывая, что имя пользователя или пароль неверны.

Клиент получает ответ от сервера и проверяет, был ли вход успешным. Если авторизация прошла успешно, интерфейс переключается на главную страницу, а данные пользователя передаются в контроллер главной страницы. Если вход не удался, пользователю отображается сообщение об ошибке.

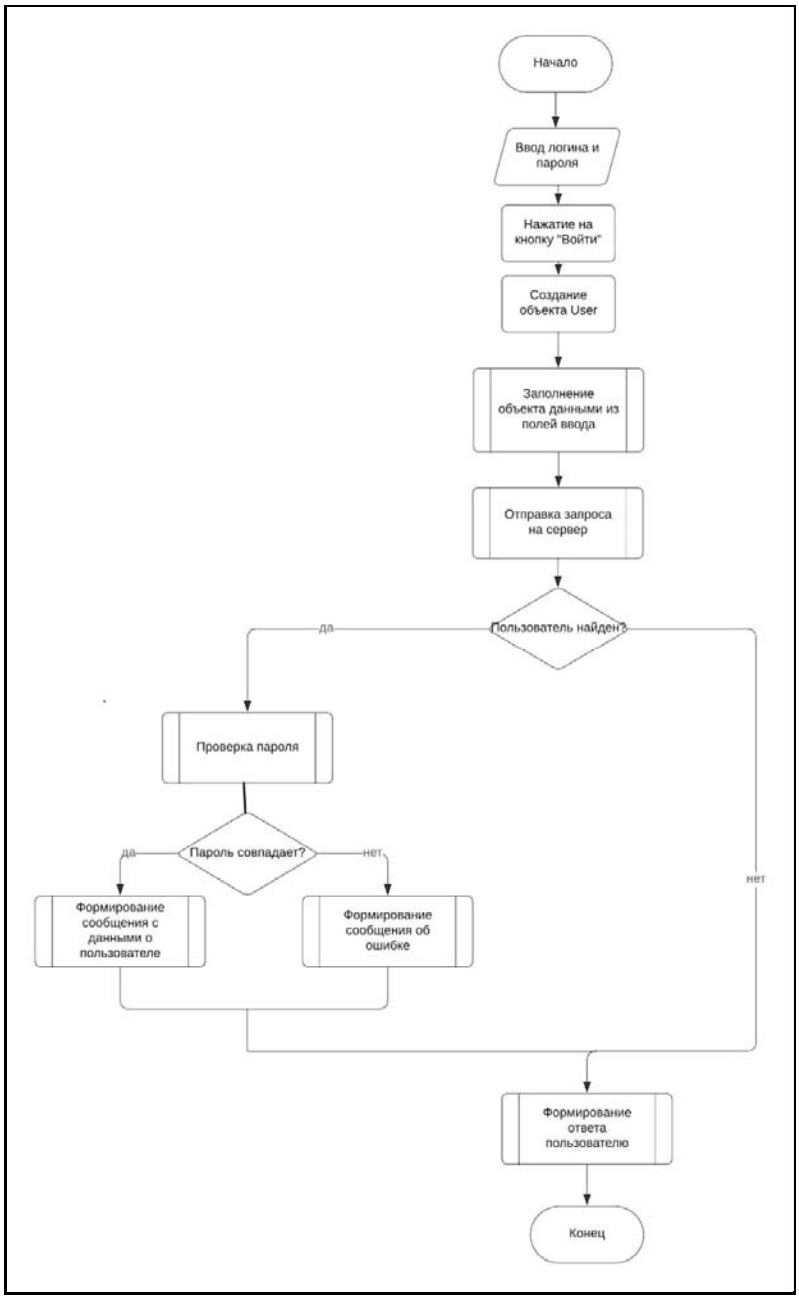


Рисунок 2.9 – Схема алгоритма авторизации

2.4 Проектирование пользовательского интерфейса

При проектировании пользовательского интерфейса системы управления персоналом акцент сделан на создании функционального и удобного пространства для пользователей, что важно для эффективного выполнения их задач.

Для реализации интерфейса использовалась библиотека JavaFX, которая позволяет создавать графические пользовательские интерфейсы с разнообразными элементами управления. Важной частью интерфейса стали таблицы, которые используются для отображения различных типов данных, включая задачи, рабочее время и другие значимые сведения. Таблицы обеспечивают структурированное представление информации, что способствует легкости её восприятия и управлению. Пользователи могут легко просматривать, сортировать и фильтровать данные, что улучшает взаимодействие с системой.

Элементы управления, такие как кнопки и метки, были тщательно продуманы для обеспечения интуитивно понятного взаимодействия. Кнопки позволяют пользователям быстро выполнять действия, такие как обновление данных или навигация по интерфейсу. Метки предоставляют важную информацию о текущем состоянии системы и пользователе, что помогает сохранять контекст работы.

Графические элементы, такие как иконки и графики, используются для повышения визуальной привлекательности интерфейса и упрощения навигации. Например, графики позволяют наглядно представлять данные о продолжительности работы, что облегчает анализ производительности.

JavaFX поддерживает адаптивный дизайн, что позволяет интерфейсу корректно отображаться на различных устройствах и экранах. Это обеспечивает удобство работы как на настольных компьютерах, так и на мобильных устройствах, что является важным аспектом в современных приложениях.

Кроме того, интерфейс обеспечивает мгновенную обратную связь, позволяя пользователям видеть результаты своих действий в реальном времени. Это достигается за счет интеграции с потоками данных, что улучшает общий пользовательский опыт и делает приложение более отзывчивым.

Таким образом, проектирование пользовательского интерфейса основывается на принципах удобства, доступности и функциональности. Использование JavaFX и его компонентов способствует созданию эффективного приложения, которое отвечает потребностям пользователей и улучшает их производительность.

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

Для разработки системы управления персоналом был выбран язык программирования Java, который обеспечивает высокую производительность, надежность и кроссплатформенность. Архитектура приложения основана на клиент-серверном взаимодействии, что позволяет эффективно разделить обязанности между компонентами системы, упрощая процесс разработки и поддержки.

Для визуализации аналитической информации, такой как графики продолжительности работы сотрудников, использовалась библиотека JavaFX. Она предоставляет инструменты для создания динамичных и интерактивных графиков и элементов управления. JavaFX поддерживает современные подходы к разработке пользовательского интерфейса, что делает приложение более интуитивно понятным и удобным для пользователей. Библиотека позволяет легко отображать данные в наглядной форме и поддерживает плавные анимации, что улучшает восприятие информации.

Для работы с данными и их обмена между клиентом и сервером использовалась библиотека Gson. Она облегчает процесс сериализации и десериализации объектов в формат JSON, что является стандартом для обмена данными в веб-приложениях. Gson позволяет быстро и эффективно конвертировать данные, что упрощает взаимодействие между различными компонентами системы.

В качестве системы управления базами данных была выбрана MySQL. Эта система управления базами данных поддерживает сложные запросы и транзакции, что критично для обеспечения корректной работы приложения в банковской сфере.

Для разработки было выбрано IDE IntelliJ IDEA, которое предлагает широкий набор инструментов для написания, отладки и анализа кода, что значительно ускоряет процесс разработки и повышает качество создаваемого программного обеспечения.

Для проектирования и визуализации архитектуры системы использовались инструменты, такие как Enterprise Architect. Эти средства позволяют создавать UML-диаграммы, что упрощает понимание структуры и взаимодействия компонентов системы.

Таким образом, выбор компонентов и технологий для реализации программного средства основан на необходимости создания надежного, эффективного и легко поддерживаемого приложения. Использование Java, JavaFX, Gson и MySQL обеспечивает прочную основу для разработки системы управления персоналом, а современные инструменты разработки способствуют созданию удобного пользовательского интерфейса и эффективной бизнес-логики.

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

После запуска программы для получения основных функций пользователю необходимо авторизоваться введя свой логин и пароль (рисунок 3.1).

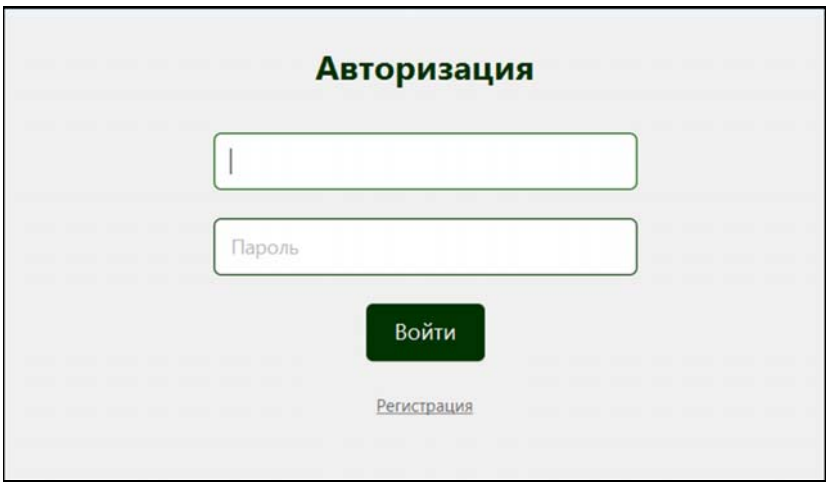


Рисунок 3.1 – Страница авторизации

В случае, если пользователь не зарегистрирован, ему необходимо нажать на кнопку “Регистрация”, после чего он сможет создать аккаунт. При попытке ввода неверных логина или пароля программа выдаст сообщение об ошибке (рисунок 3.2).

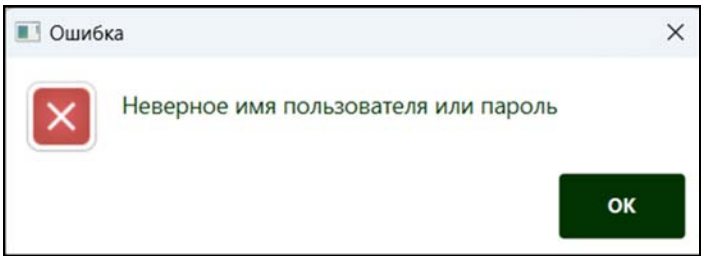


Рисунок 3.2 – Окно ошибки при неверном вводе данных пользователя

Для регистрации пользователю необходимо нажать “Регистрация”, после чего он перейдёт на страницу регистрации (рисунок 3.3).

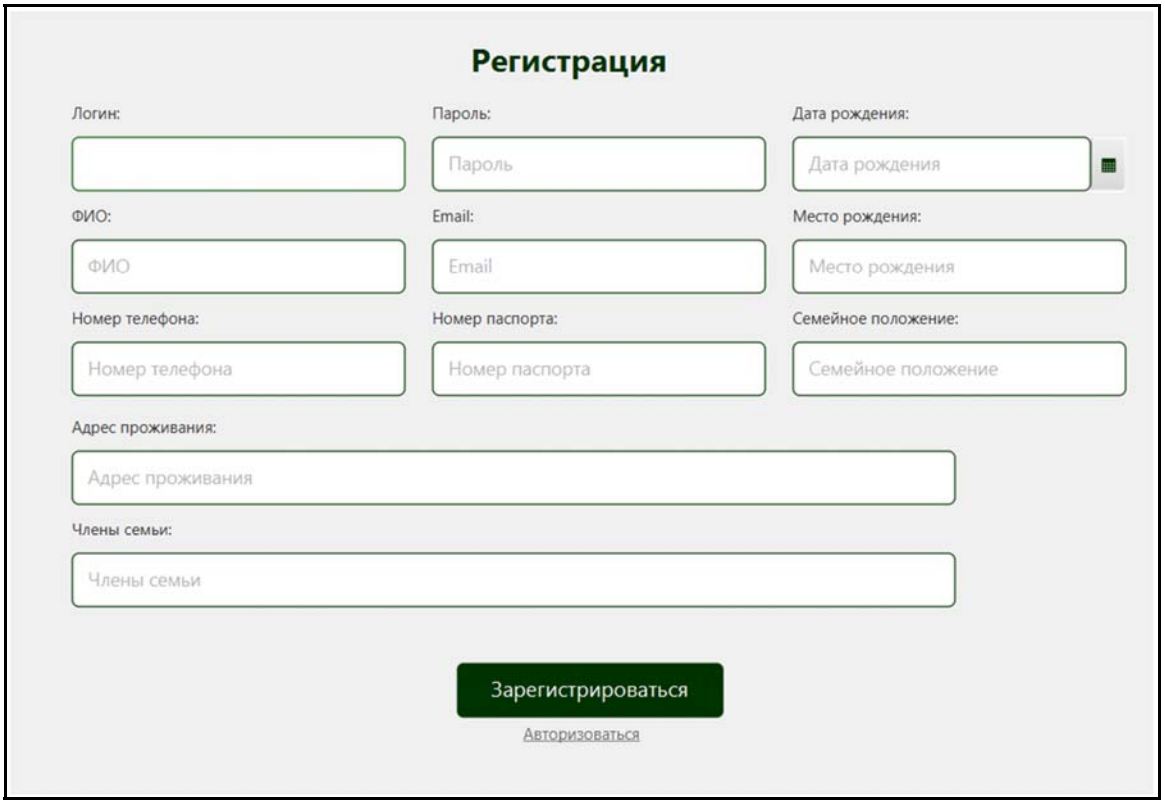


Рисунок 3.3 – Страница регистрации

При регистрации пользователю необходимо придумать пароль, который должен содержать не менее 8 символов, иметь хотя бы одну строчную и одну заглавную букву латинского алфавита, а также содержать не менее одной цифры. Логин пользователя должен содержать от 3 до 20 символов и состоять только из цифр и букв латинского алфавита. Остальные поля не должны оставаться пустыми. При неверном вводе данных для регистрации пользователю будет показано окно с информацией об ошибке. На рисунке 3.4 показано окно ошибки при пароле, который не соответствует требованиям, а именно содержит менее 8 символов.

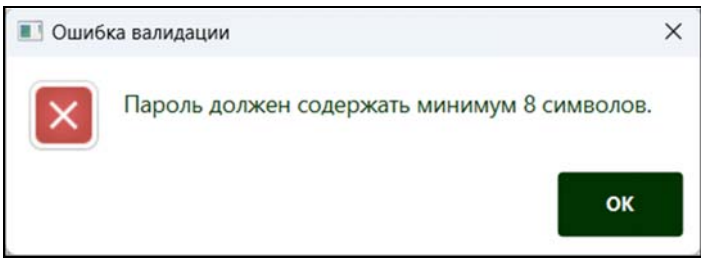


Рисунок 3.4 – Окно ошибки при неверном пароле

В поле для ФИО можно вводить строки содержащие только буквы. В случае неправильного ввода будет показано окно ошибки (рисунок 3.5).

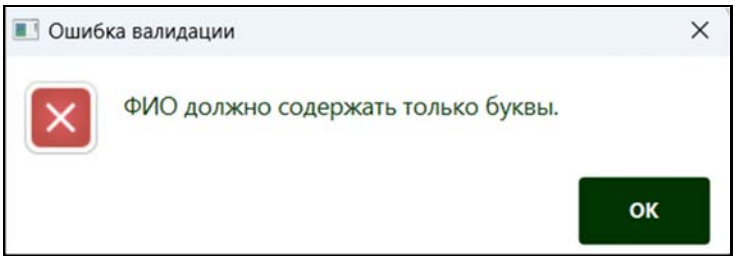


Рисунок 3.5 – Окно ошибки при неправильно введённом ФИО

Предусмотрен ввод только цифр в случае, если необходимо ввести оценку или другие данные, которые должны содержать только числовые значения. Если числа должны быть в определённом диапазоне, пользователь должен выбрать число из выпадающего списка. На рисунке 3.6 показан выпадающий список для ввода оценки качества работы пользователя. Данный список содержит только числа, при этом диапазон чисел от одного до десяти, так как оценка качества работы предполагает десятибалльную систему оценивания.

Рисунок 3.6 – Поле для ввода цены

Предусмотрен ввод только даты в полях, где необходимо её вводить. Дата, как и некоторые поля для ввода чисел, может иметь ограничения по диапазону. На рисунке 3.7 представлено поле для выбора даты задачи. Выбрать можно только дату, при этом она должна быть в диапазоне от сегодняшнего дня и позже.

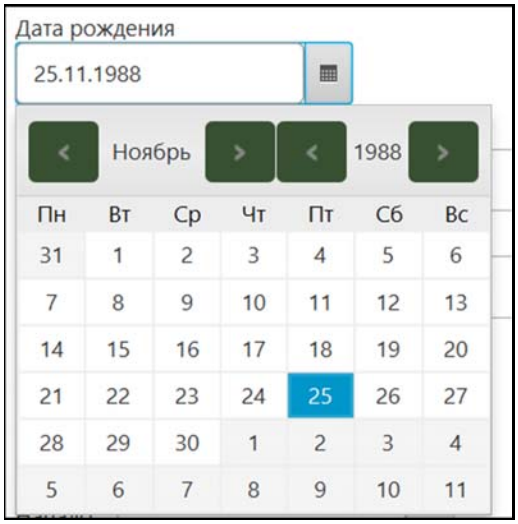


Рисунок 3.7 – Поле для ввода даты

Если пользователь не вводит никакие значения в обязательные поля, ему будет выведено сообщение об ошибке (рисунок 3.8).

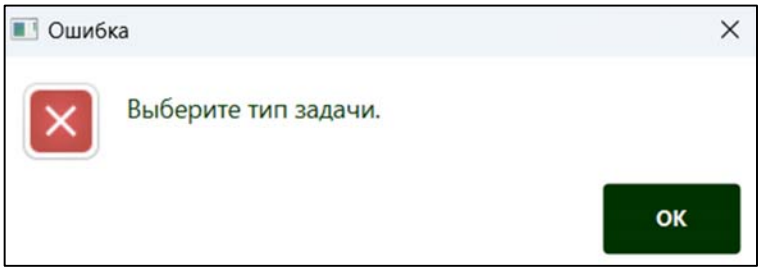


Рисунок 3.8 – Окно с ошибкой при пустом обязательном для заполнения поле

При нажатии на кнопку, которая предполагает выполнение важных действий, таких как принятие соискателя на работу, пользователю предлагается подтвердить своё решение (рисунок 3.9).

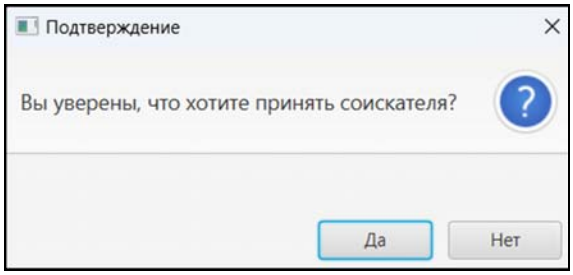


Рисунок 3.9 – Окно с подтверждением принятия соискателя на работу

После выполнения важных функций пользователю будет показано окно с информацией об успешном выполнении операции. На рисунке 3.10 показано окно с информацией об успешной регистрации.

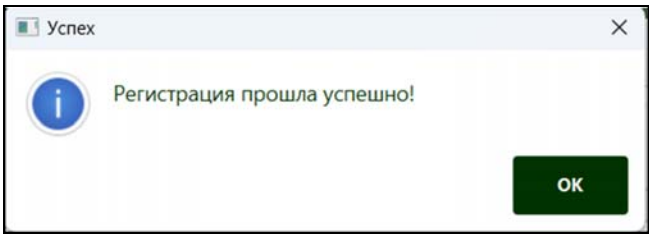


Рисунок 3.10 – Окно с информацией об успешной регистрации

В зависимости от роли, пользователям доступны различные функции и страницы. Например, если пользователь является сотрудником, после авторизации ему отображается главная страница. Если пользователь является соискателем, ему недоступна главная страница, и отображается страница соискателя. При этом, даже если пользователи с различными ролями имеют доступ к одной странице, им могут быть доступны разные функции. Например, на главной странице для пользователей с разными ролями отображаются разные названия для страниц уникальных функций, которые являются для всех различными. Для пользователя с ролью обычного сотрудника переход на страницу с уникальными функциями не отображается. Эти решения предусмотрены для безопасного доступа к данным системы и их изменения. Далее представлена навигация для обычного пользователя (рисунок 3.11), сотрудника отдела безопасности (рисунок 3.12), сотрудника отдела кадров (рисунок 3.13), руководителя отдела (рисунок 3.14) и администратора (рисунок 3.15).

Рисунок 3.11 – Панель навигации обычного сотрудника

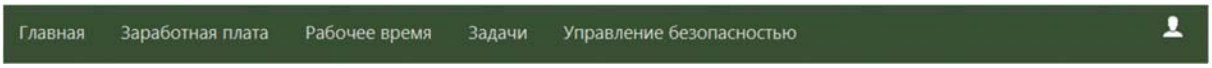


Рисунок 3.12 – Панель навигации сотрудника отдела безопасности

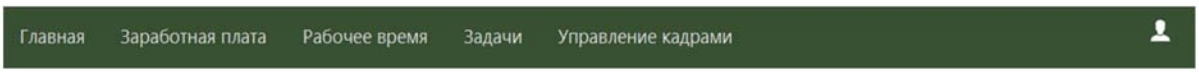


Рисунок 3.13 – Панель навигации сотрудника отдела кадров

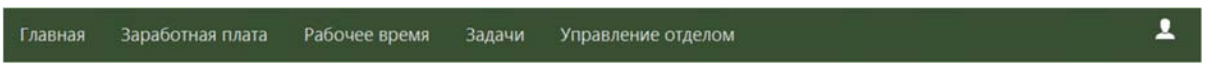


Рисунок 3.14 – Панель навигации руководителя отдела

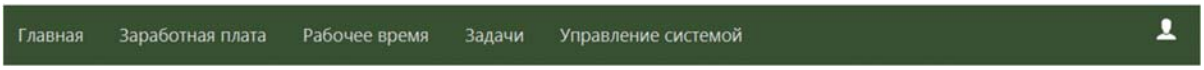


Рисунок 3.15 – Панель навигации администратора

Кроме того, сотрудники и соискатели имеют различный доступ к странице профиля. Соискатели на своей странице профиля могут только изменять и просматривать свои данные (рисунок 3.15), в то время как сотрудники могут просматривать информацию об отпусках и больничных, а также отправлять заявки на отпуск (рисунок 3.16).

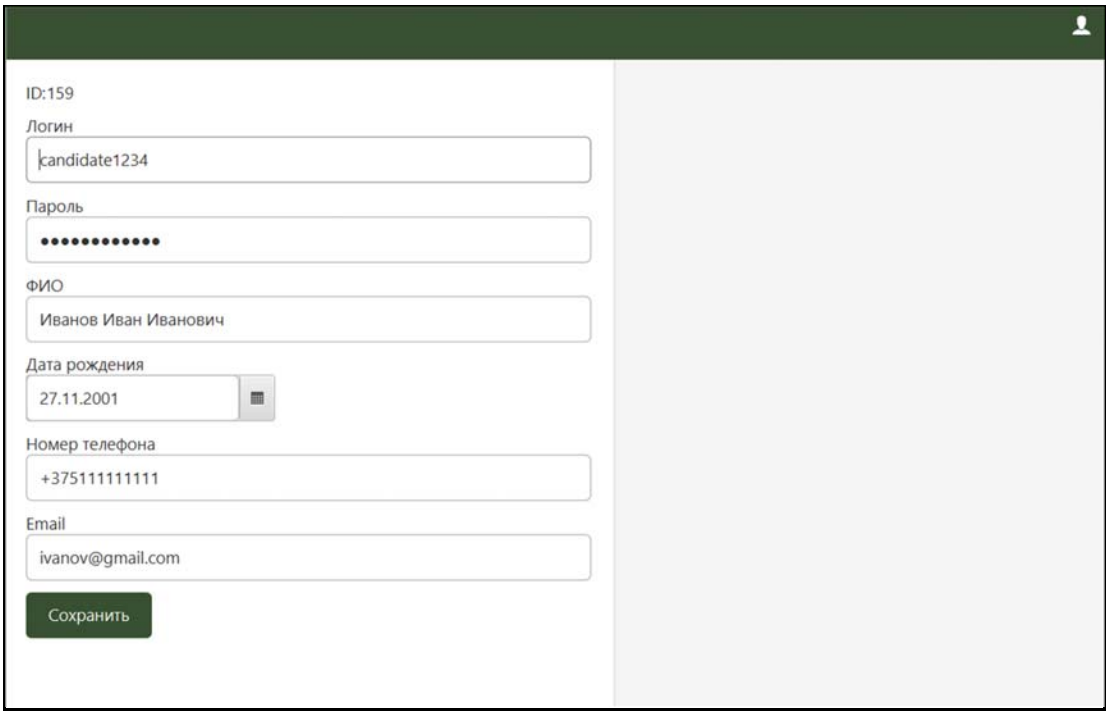


Рисунок 3.15 – Окно профиля соискателя

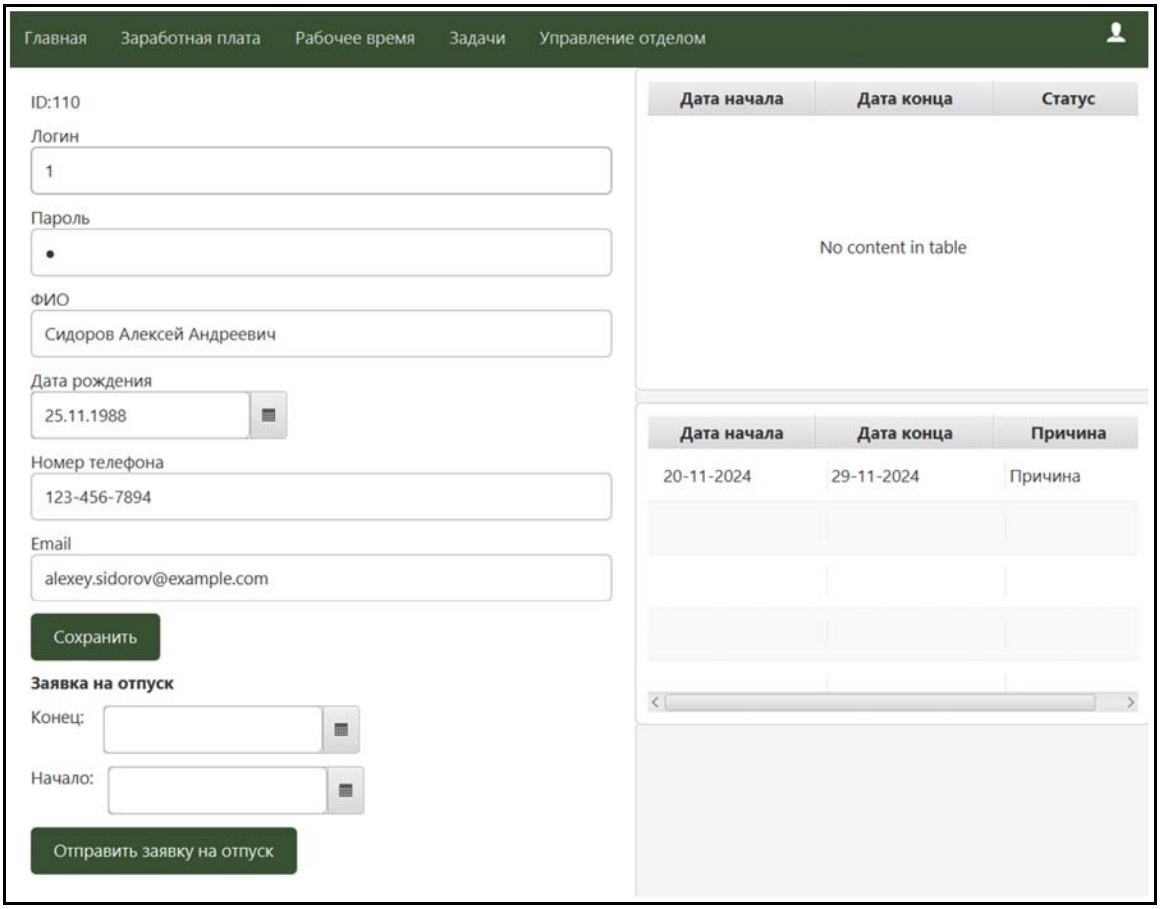


Рисунок 3.16 – Окно профиля сотрудника

4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

Для корректного функционирования приложения необходимо наличие таких компонентов, как IntelliJ IDEA и MySQL Workbench. Перед первым запуском приложения необходимо установить сервер баз данных MySQL, если он ещё не установлен, и скачать MySQL Workbench. Затем создайте новую схему базы данных с названием «personnel_management». Для запуска приложения требуется установить JDK 21. После установки откройте серверную часть проекта в IntelliJ IDEA и выберите JDK 21 в разделе Project Structure -> Project -> SDK. Затем запустите приложение, нажав кнопку «Run».

В данной программе поддерживается шесть уровней доступа к информации: администратор, обычный сотрудник, сотрудник отдела кадров, сотрудник отдела безопасности, руководитель отдела и соискатель. После запуска программы в окне отображается главная страница авторизации, где пользователю предлагается войти в аккаунт. Если аккаунт не создан, пользователю предлагается зарегистрироваться. При вводе правильных логина и пароля пользователь видит главную страницу, если он является сотрудником (рисунок 4.1), или страницу соискателя, если он является соискателем (рисунок 4.2).

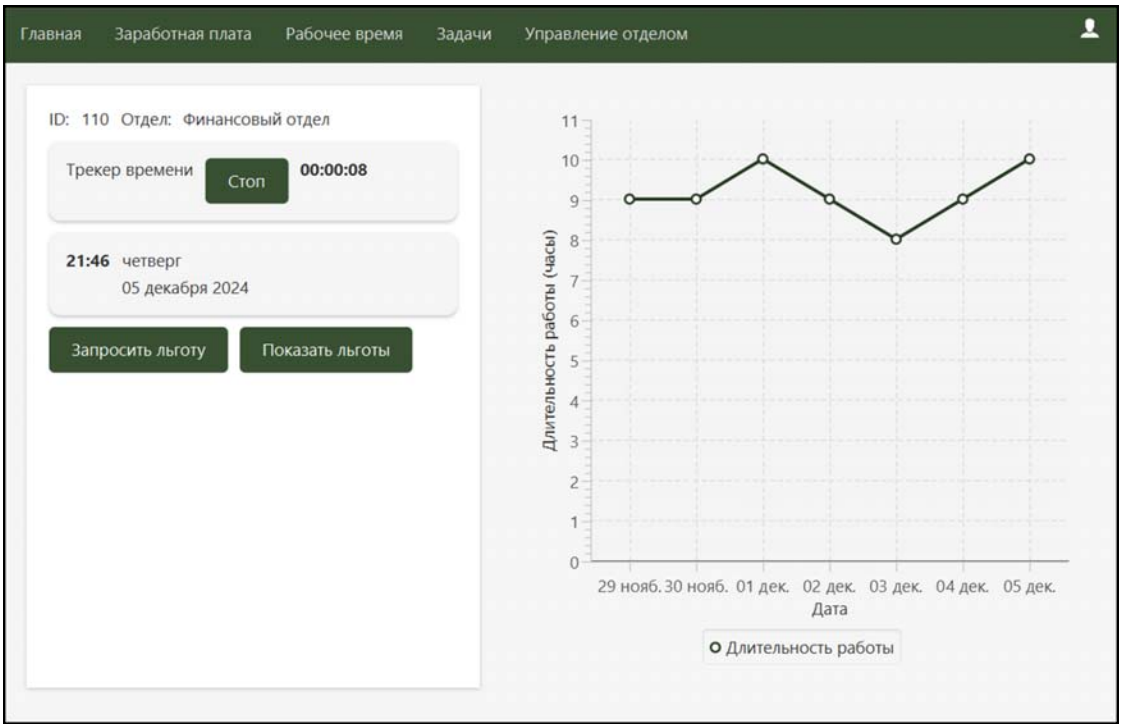


Рисунок 4.1 – Главная страница

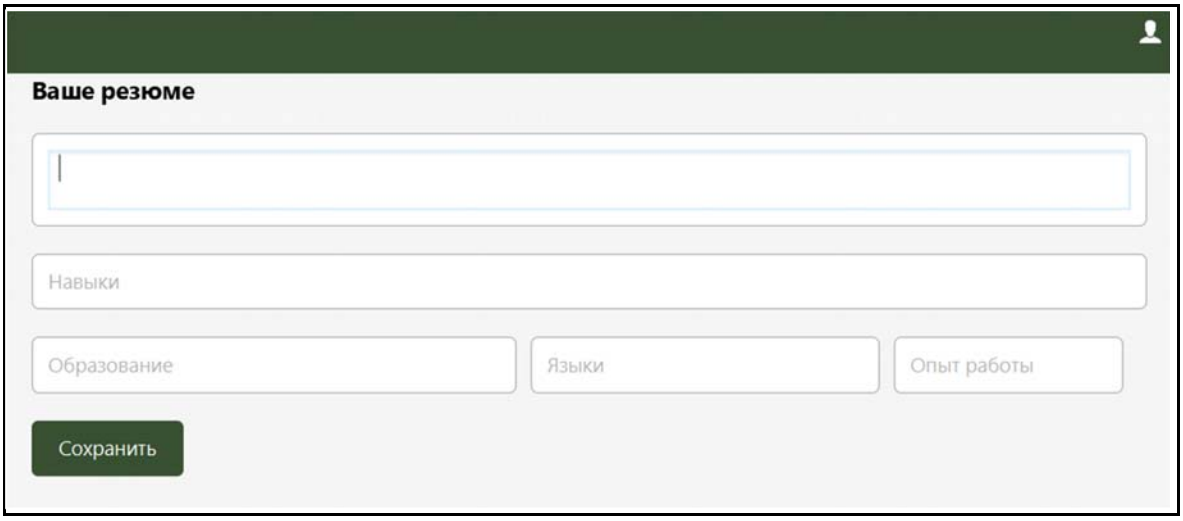


Рисунок 4.2 – Страница соискателя

При входе от имени соискателя пользователь может отправить резюме. После отправки информация будет отображаться в полях для ввода данных резюме. При вводе другой информации пользователь может нажать на кнопку “Подтвердить”, и таким образом информация о резюме будет изменена (рисунок 4.3). После проверки информации о пользователе сотрудником отдела безопасности и одобрения резюме сотрудником отдела кадров, для соискателя блокируется возможность изменения резюме, и появляется область для записи на собеседование, если оно одобрено (рисунок 4.4). После отправки информации о записи пользователю выводится информация о собеседовании, и область с записью перестает отображаться (рисунок 4.5). Если запись одобрит сотрудник отдела кадров, запись примет статус “Запланировано”. Если запись не будет одобрена менее чем за 2 дня до назначенной даты, запись автоматически будет отменена, и пользователю будет снова предоставлена возможность записаться (рисунок 4.6). После повторной записи предыдущая запись автоматически будет удалена.

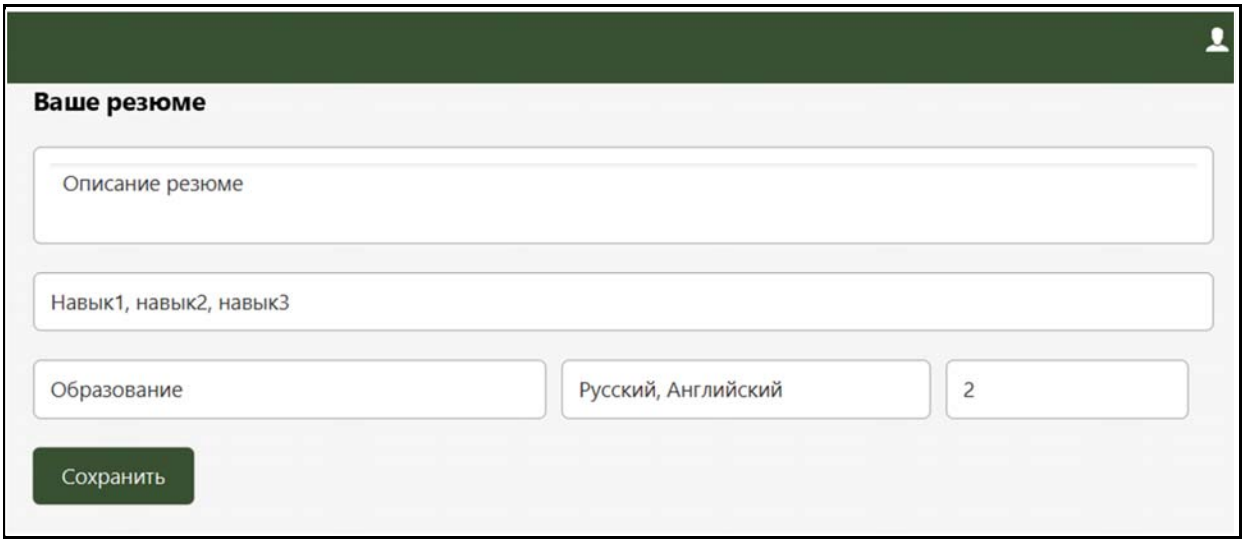


Рисунок 4.3 – Окно с отправленным резюме

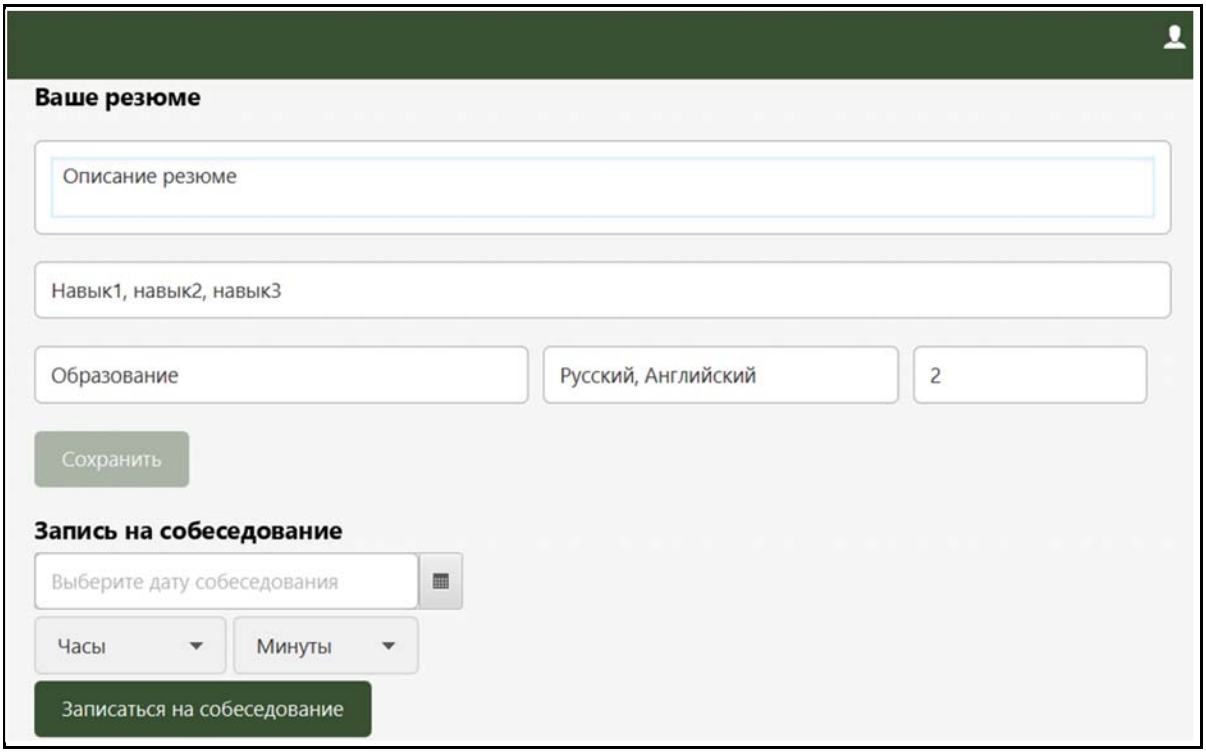


Рисунок 4.4 – Окно с одобренным резюме

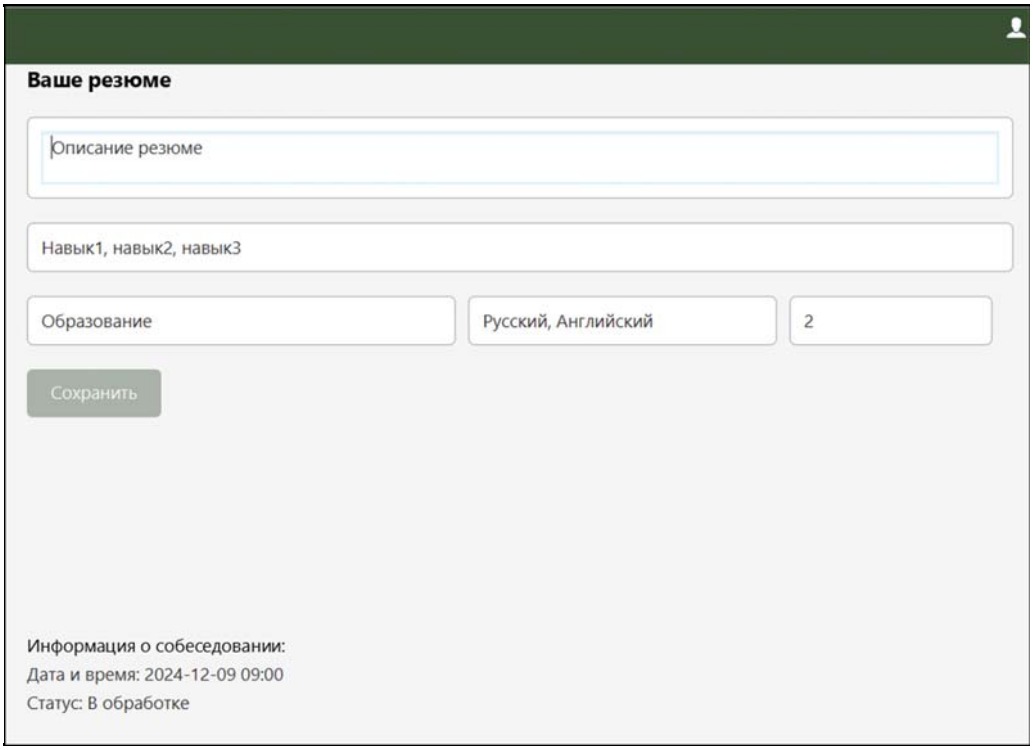


Рисунок 4.5 – Окно с информацией о записи

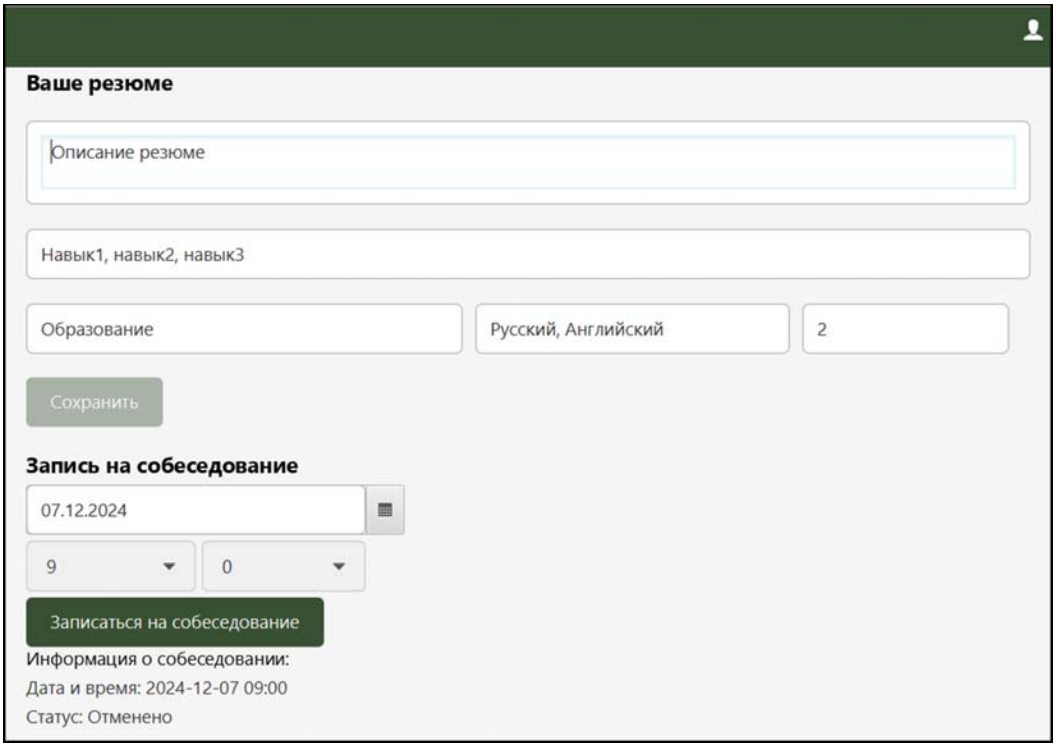


Рисунок 4.6 – Окно с отклонённым резюме

При входе от имени администратора откроется окно “Управление системой” (рисунок 4.7). На данной странице мы можем просматривать информацию о пользователях и удалять их. Также у администратора есть возможность добавлять и удалять отделы и расписания для них. Для добавления расписания необходимо ввести необходимые данные и нажать на кнопку “Добавить расписание”, а для добавления отдела — ввести его название и выбрать расписание.

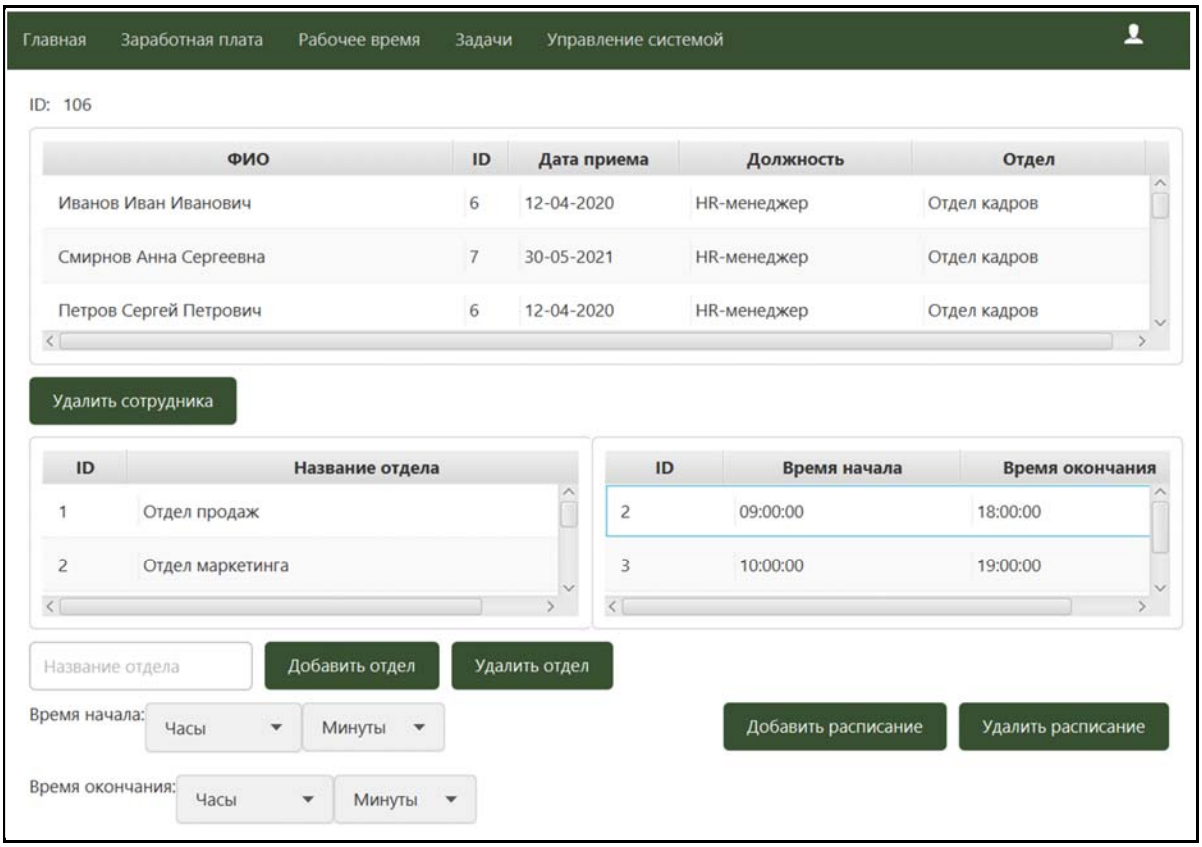


Рисунок 4.7 – Окно “Управление системой”

При входе от имени сотрудника отдела кадров доступны следующие страницы: “Отпуска”, “Больничные”, “Сотрудники”, “Соискатели”, “Заработная плата” и “Льготы”. На странице “Отпуска” сотрудник может просматривать информацию об отпусках и изменять статус заявки на отпуск (рисунок 4.8). На странице “Больничные” можно просматривать информацию о больничных листах и регистрировать их, заполнив необходимые поля, выбрав сотрудника из списка и нажав на кнопку “Добавить больничный лист” (рисунок 4.9).

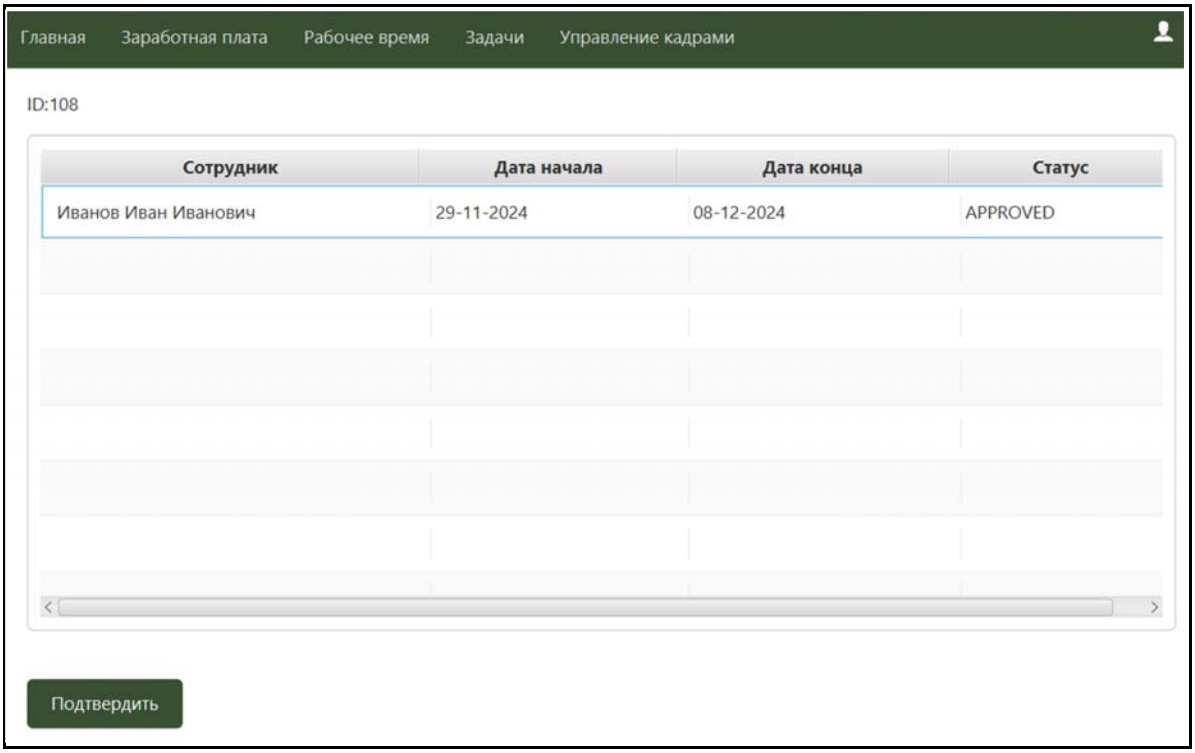


Рисунок 4.8 – Страница “Отпуска”

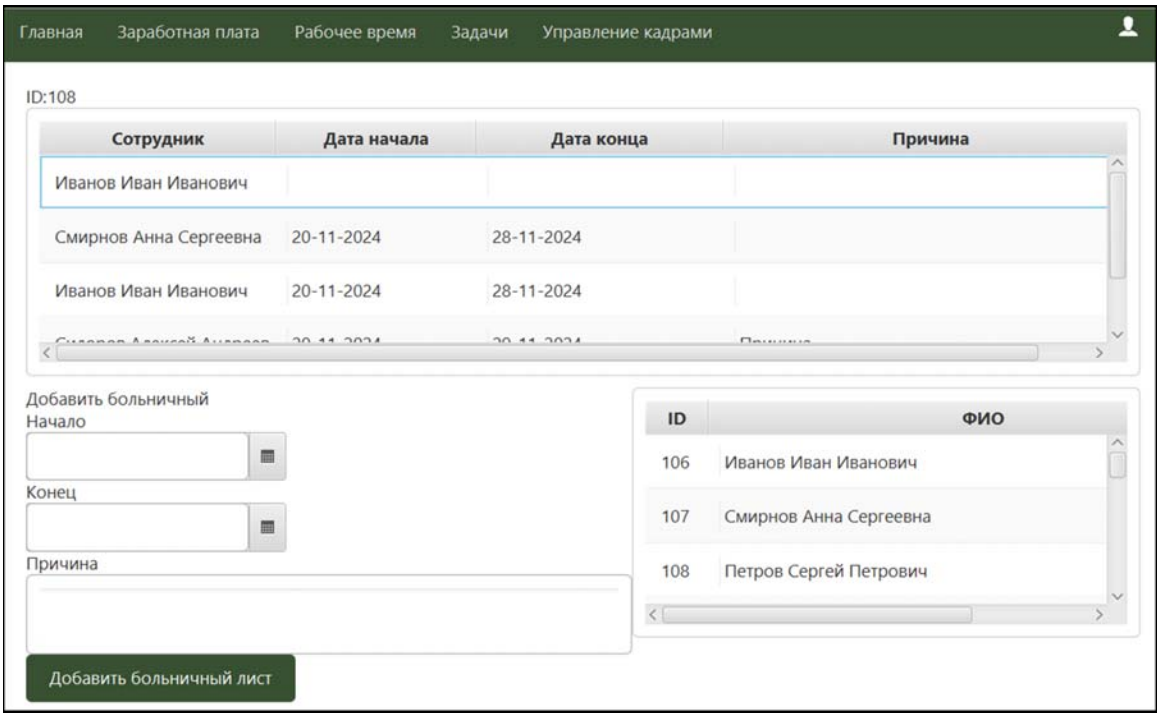


Рисунок 4.9 - Страница “Больничные”

На странице “Сотрудники” пользователь может зафиксировать информацию об увольнении и перевести сотрудника в другой отдел. Для этого ему потребуется выбрать сотрудника, отдел, роль и ввести название новой должности (рисунок 4.10).

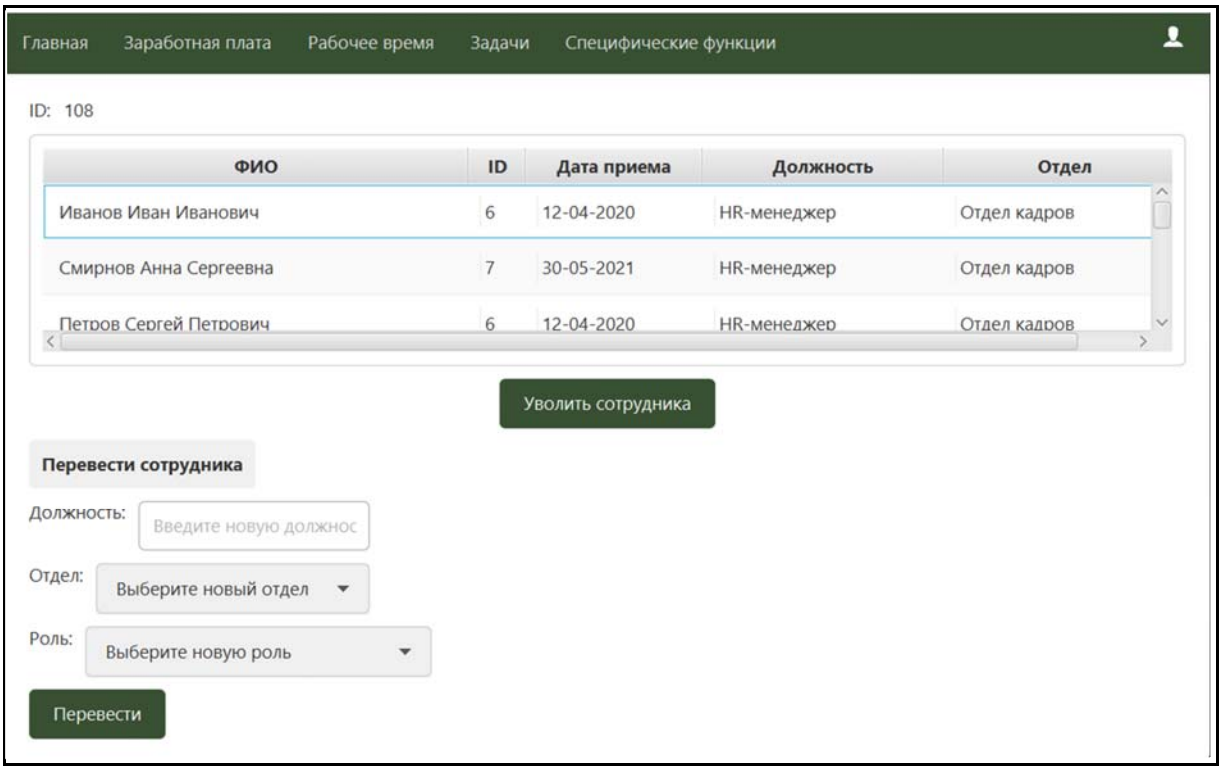


Рисунок 4.10 – Страница “Сотрудники”

На странице “Соискатели” сотрудник отдела кадров может просматривать и изменять статус резюме проверенных пользователей и записей на собеседование. Если сотрудник отдела кадров изменит статус записи на “Запланировано”, он автоматически станет интервьюером. Если он выберет статус “Завершено”, ему будет предложено принять или не принять сотрудника на работу (рисунок 4.11). Если сотрудник решит принять кандидата, ему нужно будет ввести необходимую информацию о новом сотруднике, такую как заработная плата, должность, роль и отдел (рисунок 4.12).

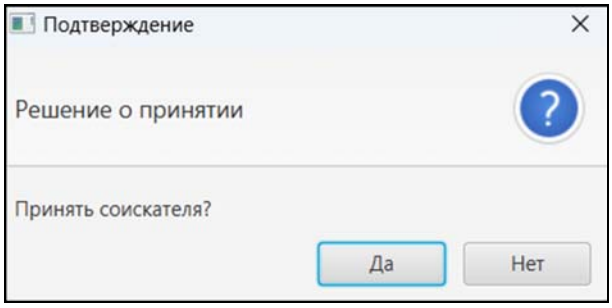


Рисунок 4.11 – Окно с выбором принятия пользователя

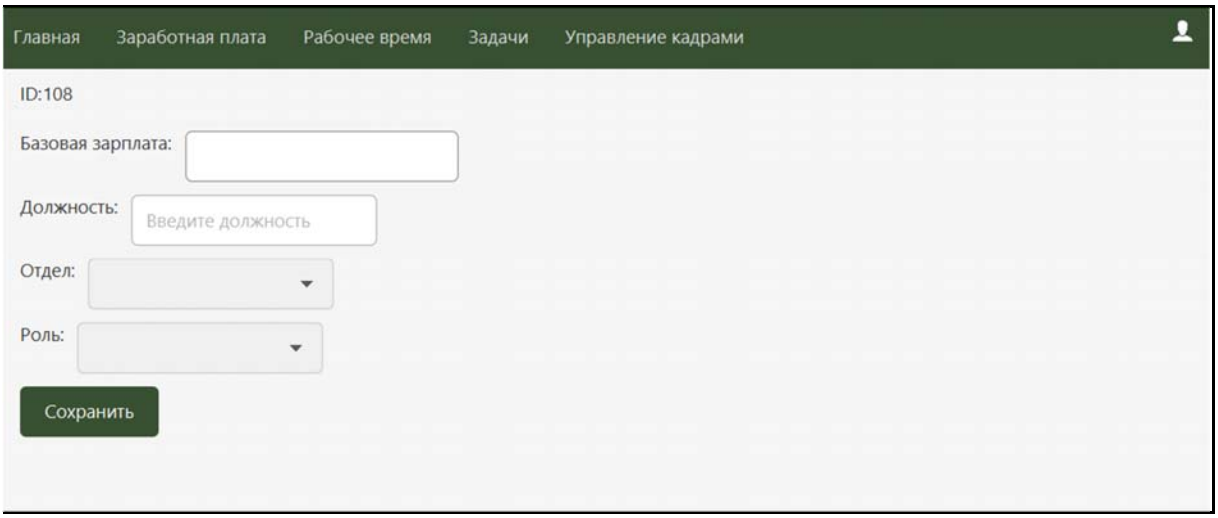


Рисунок 4.12 – Окно с вводом информации сотрудника

На странице “Заработная плата” сотрудник может вывести в таблицу данные о начисленных зарплатах при помощи кнопки “Вывести начисленные” (рисунок 4.13). Для расчёта заработной платы необходимо выбрать период начисления и дату начисления, а также изменить настройки коэффициентов для расчёта, если это необходимо (рисунок 4.14). После этого будет выведена информация о рассчитанной заработной плате (рисунок 4.15). Для сохранения информации о начислении заработной платы необходимо нажать кнопку “Начислить зарплату”. Для получения отчёта в pdf формате сотрудник должен нажать на кнопку “Получить отчёт”. Пример отчёта представлен на рисунке 4.16.

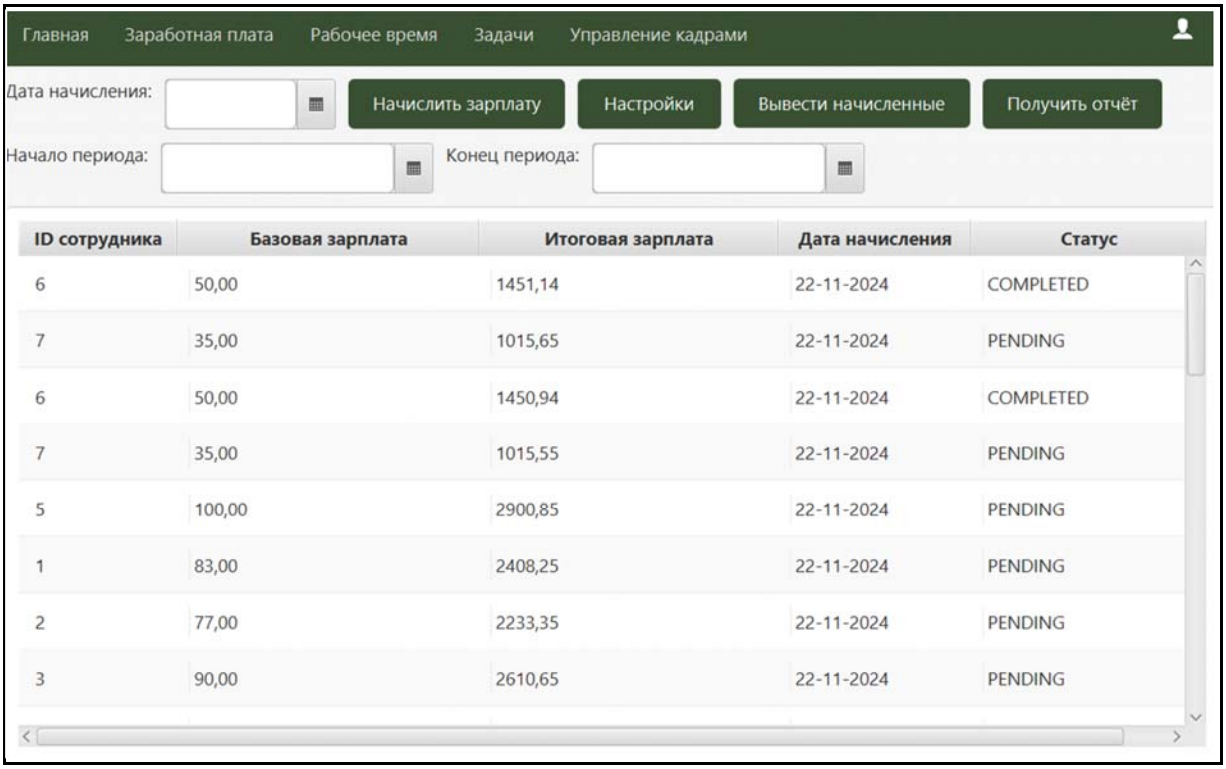


Рисунок 4.13 – Страница “Заработная плата” с информацией о начисленных зарплатах

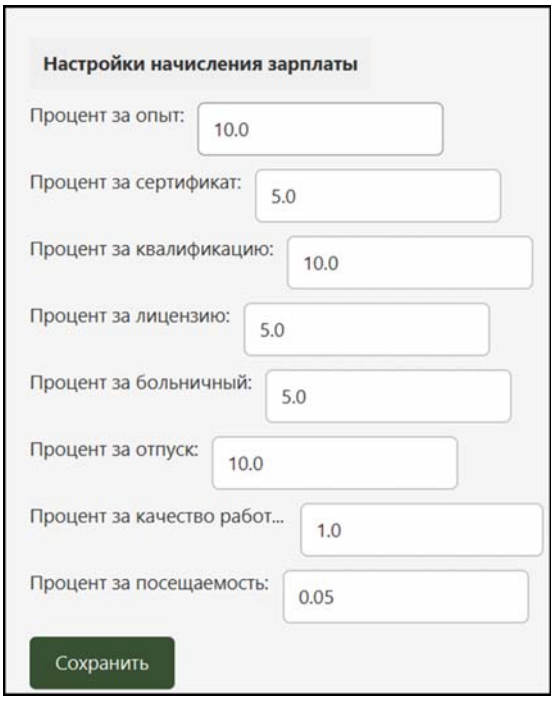


Рисунок 4.14 – Окно настройки коэффициентов

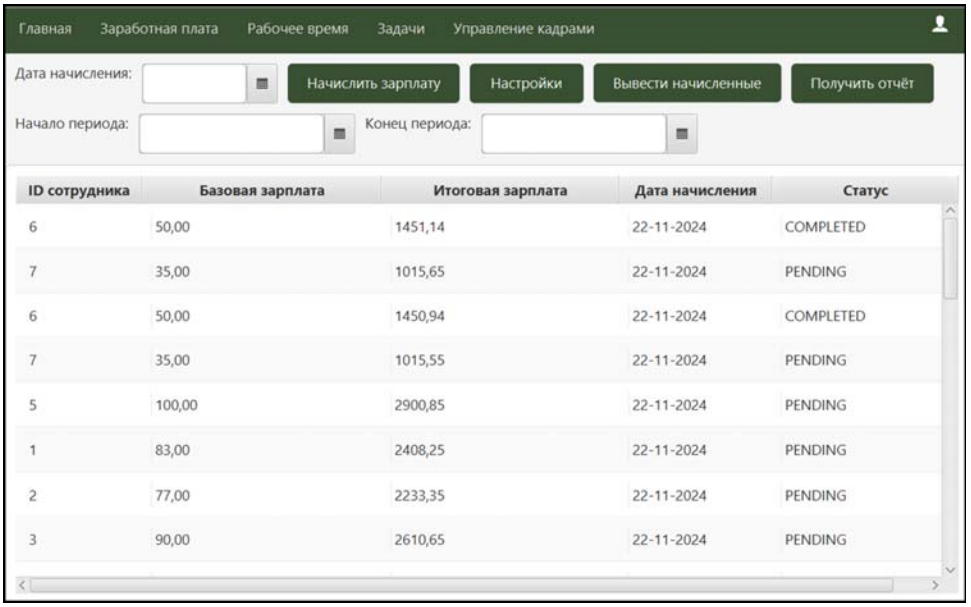
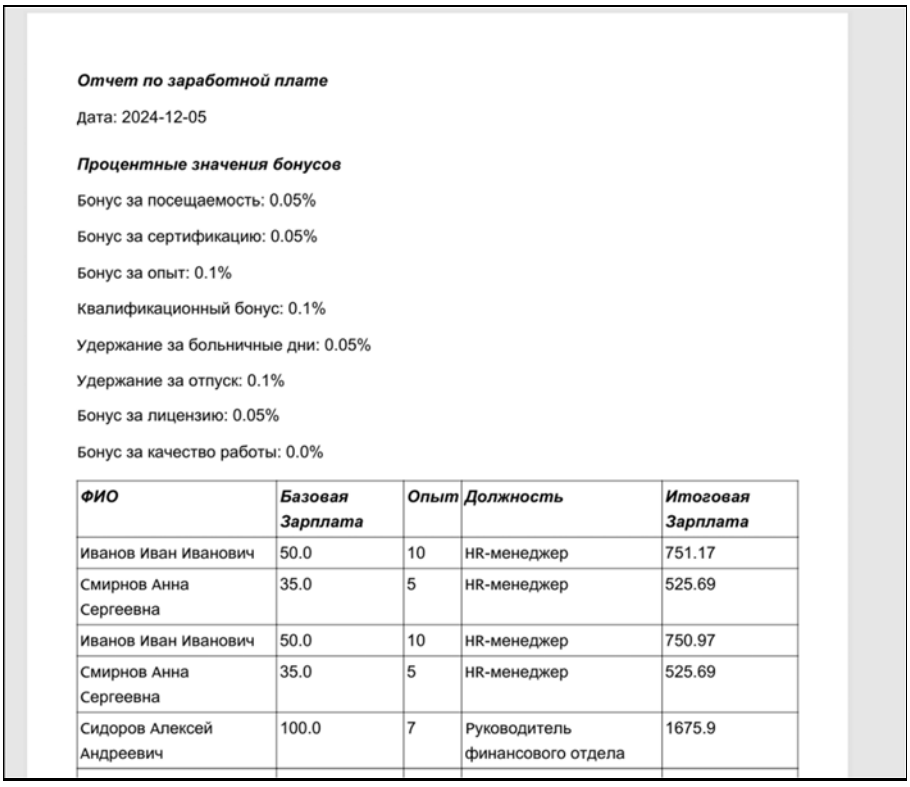


Рисунок 4.15 – Страница “Заработная плата” с информацией о рассчитанных зарплатах



Отчет по заработной плате

Дата: 2024-12-05

Процентные значения бонусов

Бонус за посещаемость: 0.05%

Бонус за сертификацию: 0.05%

Бонус за опыт: 0.1%

Квалификационный бонус: 0.1%

Удержание за больничные дни: 0.05%

Удержание за отпуск: 0.1%

Бонус за лицензию: 0.05%

Бонус за качество работы: 0.0%

ФИО	*Базовая Зарплата*	*Опыт*	*Должность*	*Итоговая Зарплата*
Иванов Иван Иванович	50.0	10	HR-менеджер	751.17
Смирнов Анна Сергеевна	35.0	5	HR-менеджер	525.69
Иванов Иван Иванович	50.0	10	HR-менеджер	750.97
Смирнов Анна Сергеевна	35.0	5	HR-менеджер	525.69
Сидоров Алексей Андреевич	100.0	7	Руководитель финансового отдела	1675.9

Рисунок 4.16 – Пример отчёта о заработной плате

На странице “Льготы” сотрудник может просматривать и изменять статус заявок на льготы.

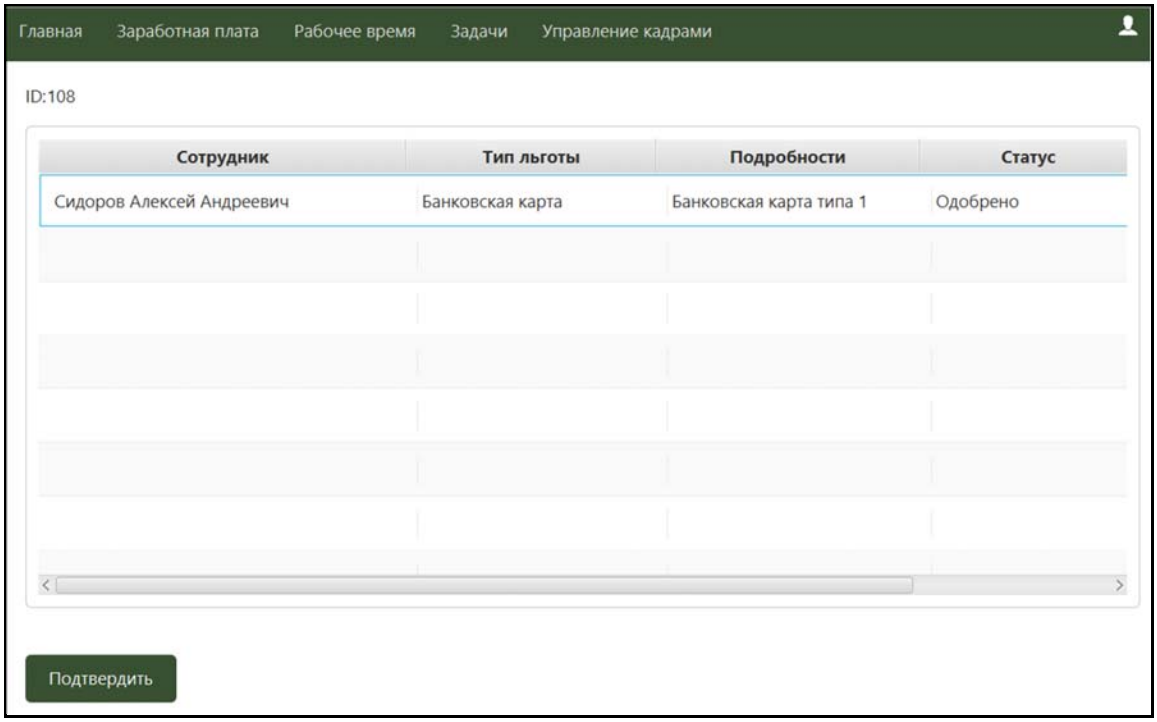


Рисунок 4.17 – Страница “Льготы”

При входе с ролью сотрудника отдела безопасности пользователю доступны страницы “Журнал инцидентов” и “Проверка данных соискателей”. На странице “Журнал инцидентов” пользователь может просматривать и добавлять новые записи об инцидентах (рисунок 4.18). Для добавления инцидента пользователю необходимо выбрать сотрудника, а также ввести информацию об инциденте (рисунок 4.19).

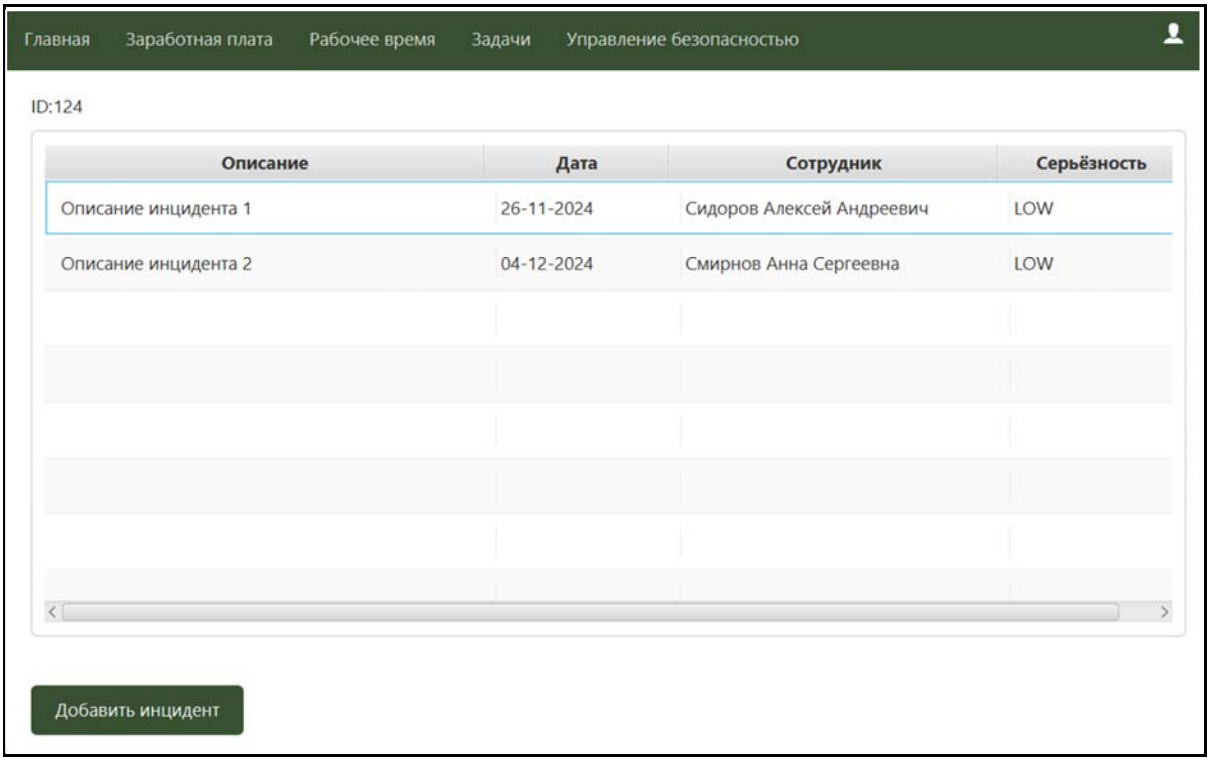


Рисунок 4.18 – Страница “Журнал инцидентов”

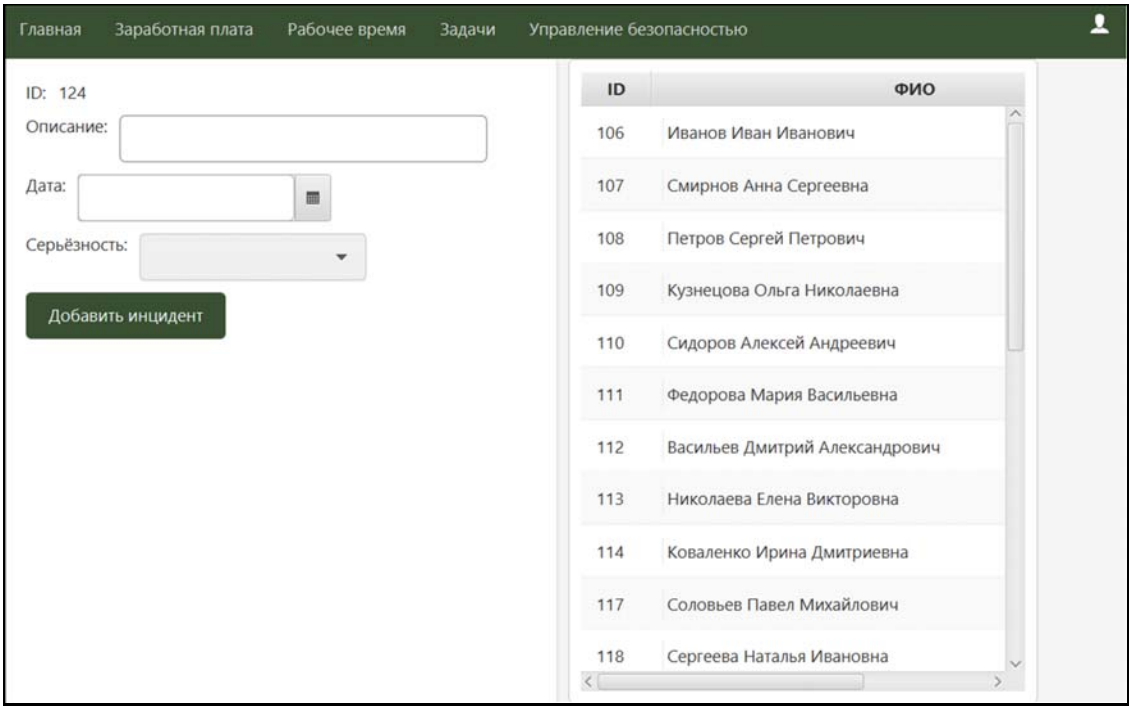


Рисунок 4.19 – Страница для добавления записи об инциденте

На странице “Проверка соискателей” сотрудник отдела безопасности может просматривать информацию, необходимую для проверки соискателя, а также изменять статус проверки (рисунок 4.20).

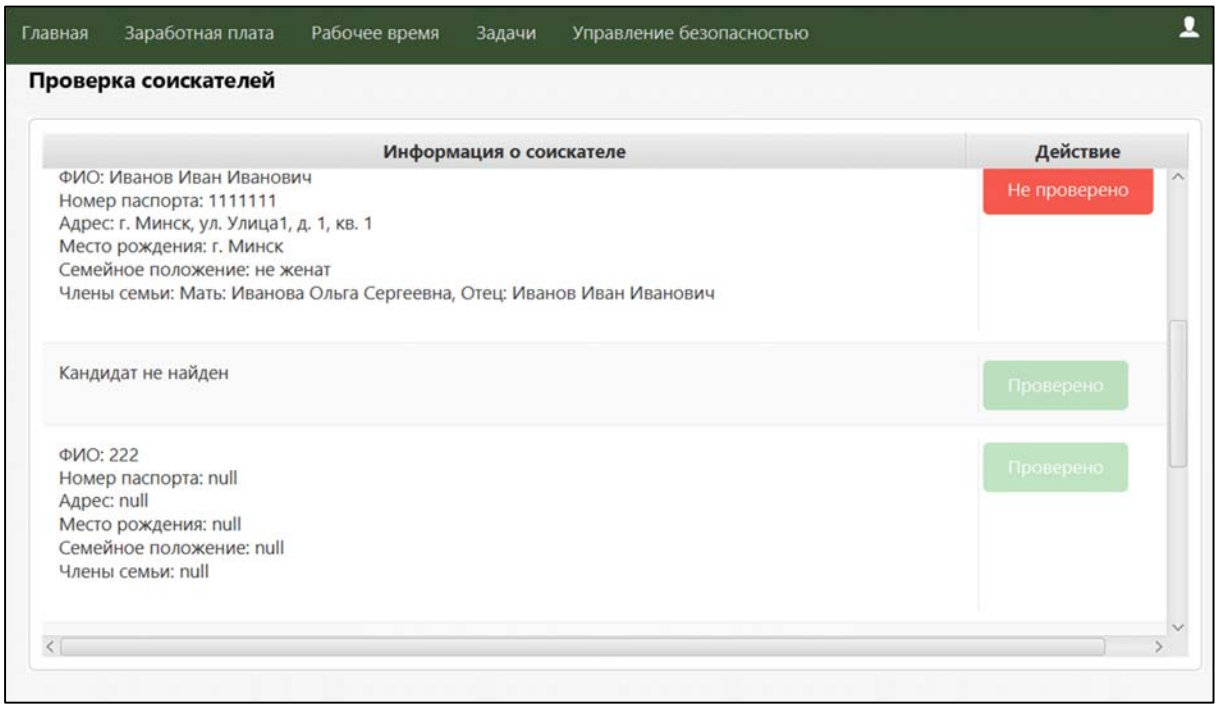


Рисунок 4.20 – Страница “Проверка соискателей”

При входе от имени руководителя отдела кадров пользователю доступны следующие страницы: “Задачи”, “Сотрудники” и “Отслеживание времени работы”. На странице “Задачи” пользователь может просматривать, назначать задачи и добавлять новые типы задач (рисунок 4.21). На рисунке 4.22 показана страница с добавлением типа задачи. Для добавления типа необходимо ввести название типа, а также выбрать документы, необходимые для выполнения этого типа задачи. Для назначения задачи необходимо выбрать тип задачи, в зависимости от которого будут отфильтрованы сотрудники; после фильтрации останутся только те сотрудники, которые имеют подходящие документы. Далее необходимо описать задачу, выбрать срок и оценку сложности задачи, после чего нажать на кнопку “Добавить задачу” (рисунок 4.23).

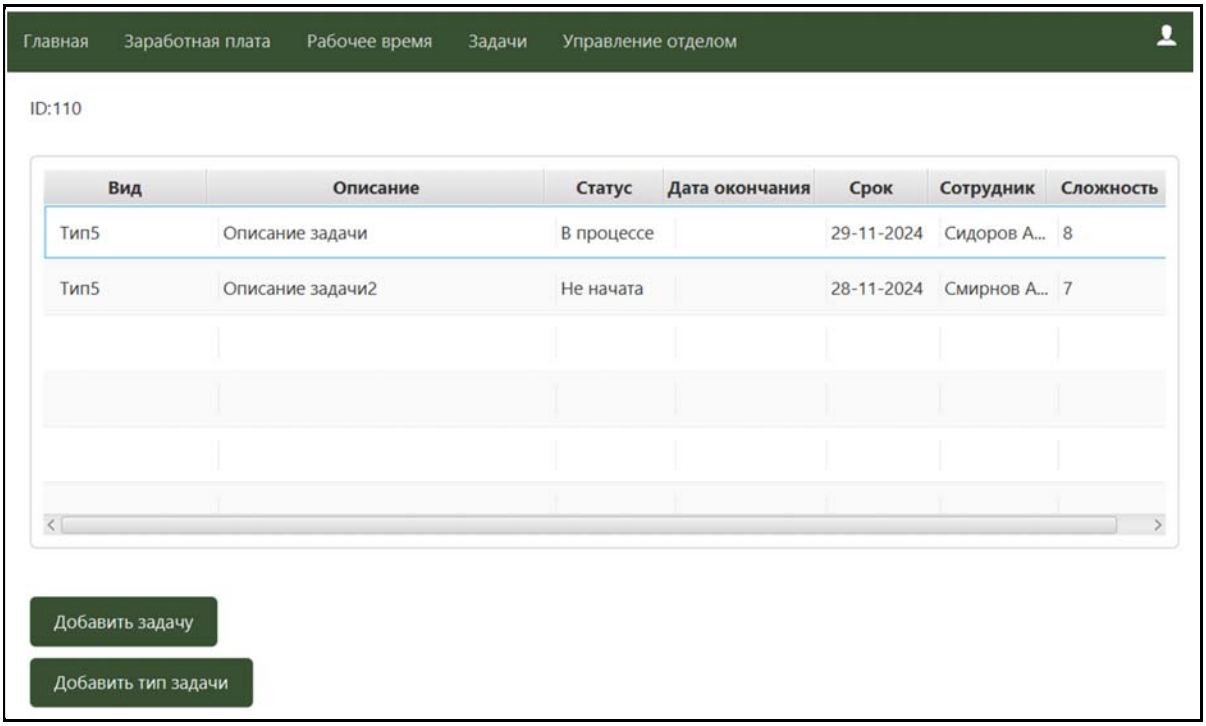


Рисунок 4.21 – Страница “Задачи”

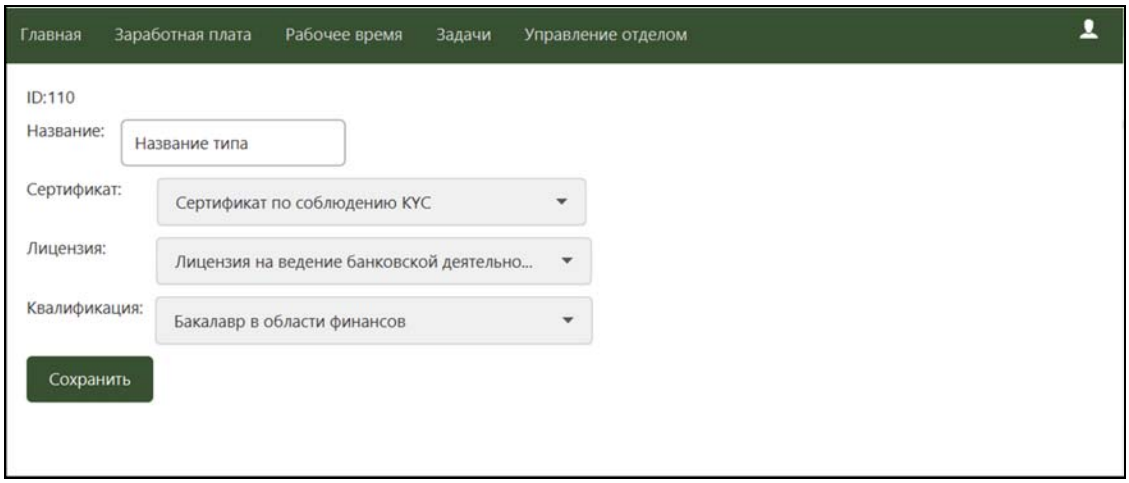


Рисунок 4.22 – Страница с добавлением типа задачи

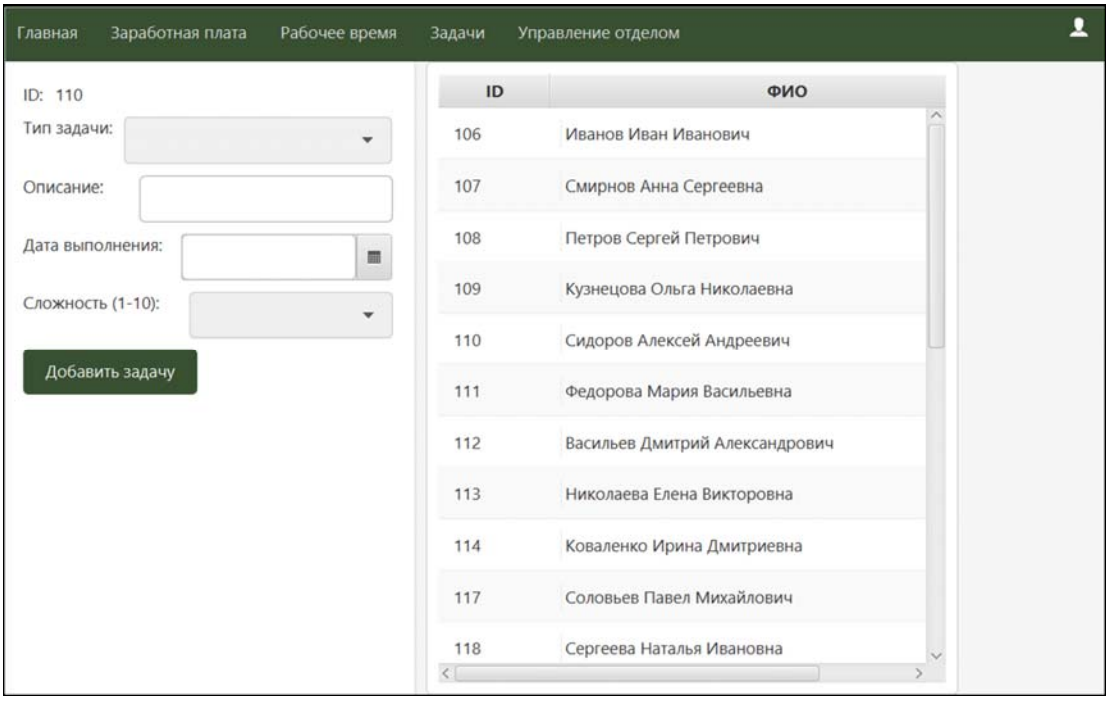


Рисунок 4.23 – Страница с добавлением задачи

На странице “Сотрудники” руководитель может просматривать список сотрудников своего отдела, добавлять документы сотрудников, просматривать и удалять лицензии и сертификаты, а также оценивать качество работы сотрудников (рисунок 4.24).

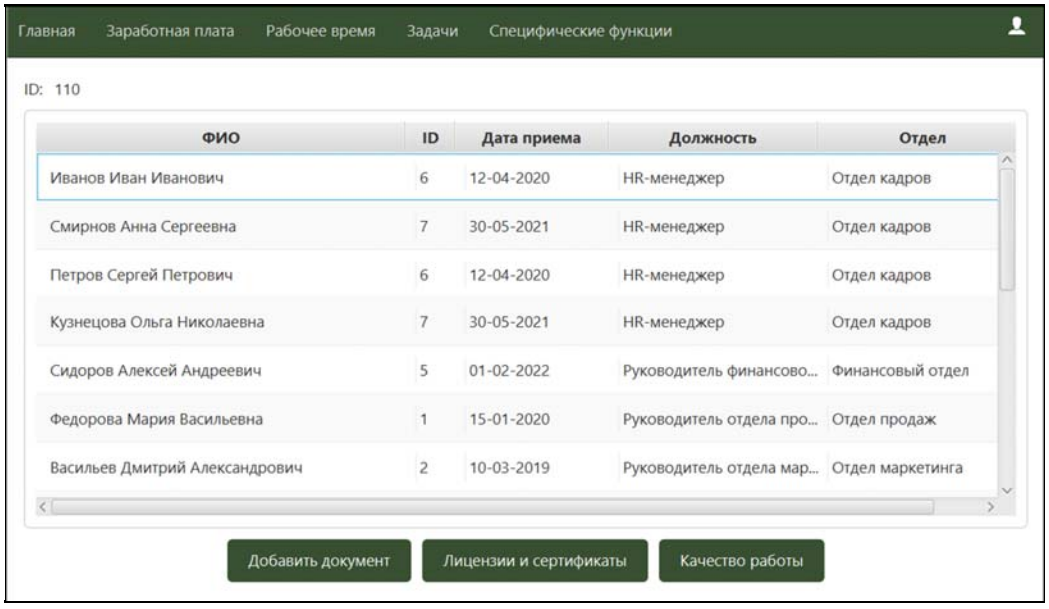


Рисунок 4.24 – Страница “Сотрудники”

Для добавления документа необходимо нажать на кнопку “Добавить документ”, после чего пользователь перейдёт на страницу с добавлением документа, где необходимо выбрать тип документа (рисунок 4.25). После выбора документа в зависимости от его типа будут представлены поля для заполнения информации о документе и выбор из списка документов данного типа. На рисунке 4.26 изображена страница для добавления документа типа “Сертификат”. Если ни один документ в списке не подходит, необходимо выбрать в списке пункт “Другое”, после чего появится дополнительное поле, куда можно будет ввести название документа.

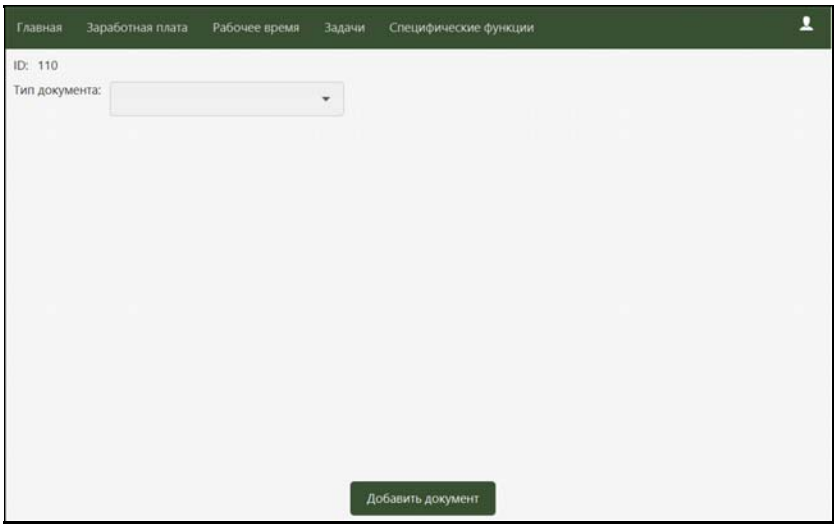


Рисунок 4.25 – Страница для добавления документа

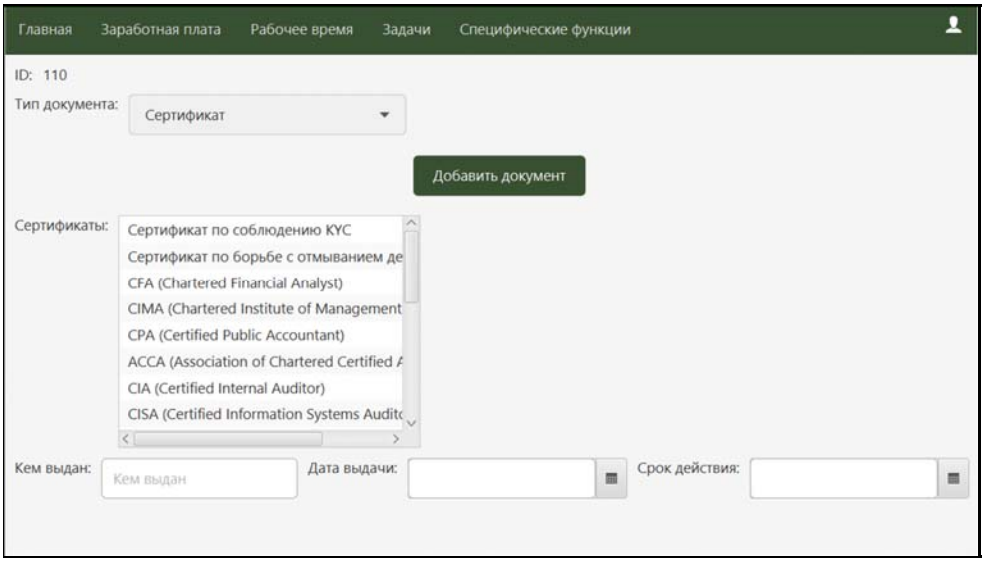


Рисунок 4.26 – Страница для добавления документа типа “Сертификат”

Для просмотра и удаления лицензий и сертификатов необходимо нажать на кнопку “Лицензии и сертификаты”, после чего пользователю будет показана страница с лицензиями и сертификатами (рисунок 4.27). Даты со сроками, которые истекли, будут подсвечены красным цветом. Для удаления документа необходимо выбрать документ для удаления в таблице и нажать на кнопку “Удалить”.

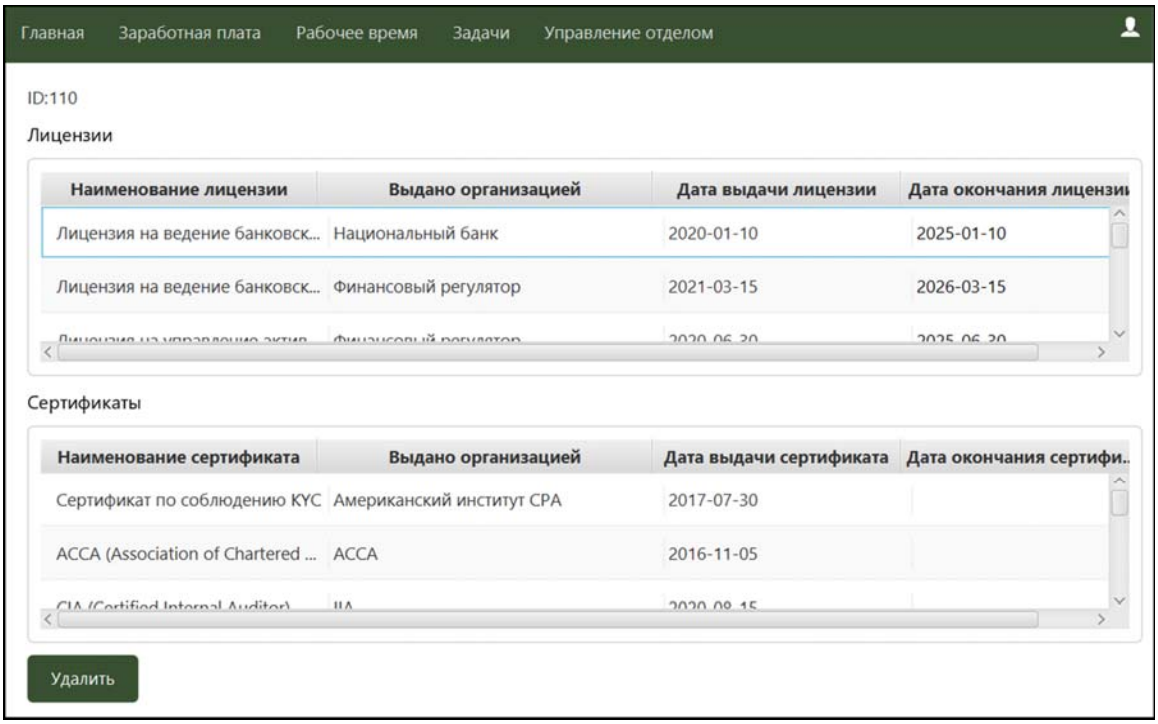


Рисунок 4.27 – Страница с лицензиями и сертификатами

Для оценки качества работы необходимо нажать на кнопку “Качество работы”, после чего отобразится страница для оценки качества работы (рисунок 4.28). На данной странице будет отображена таблица, где будут указаны ФИО, количество задач и автоматически рассчитанная оценка качества работы сотрудника.

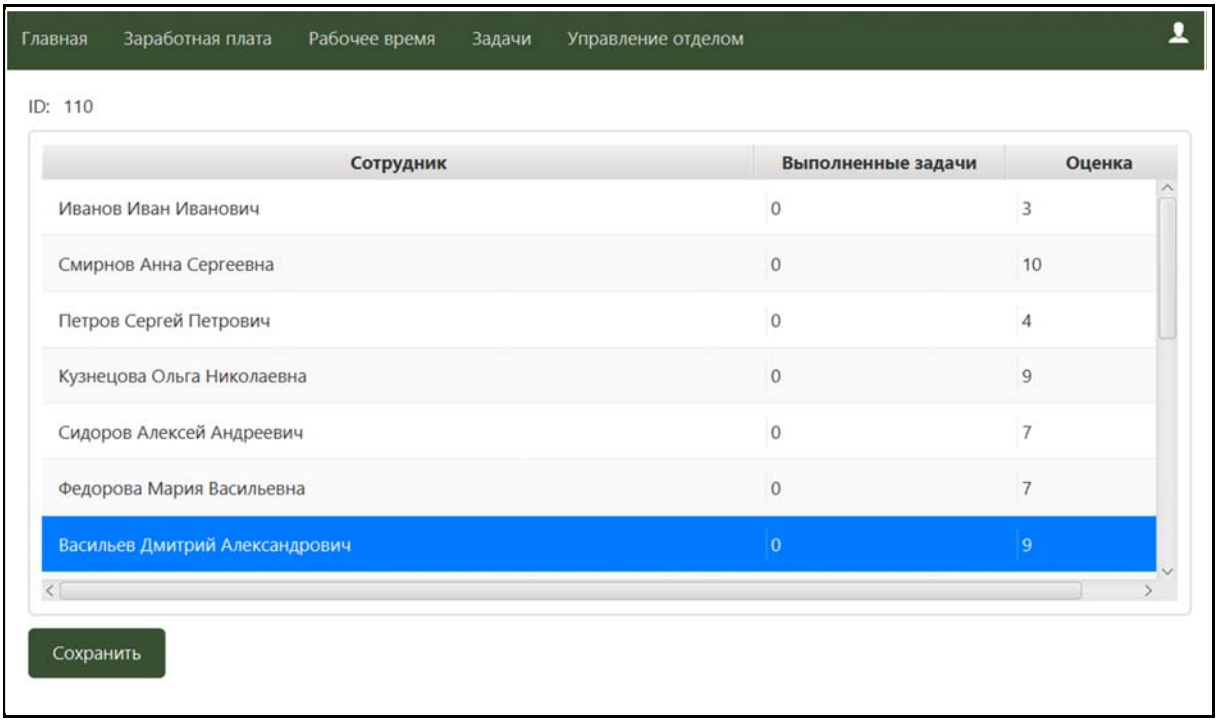


Рисунок 4.28 – Страница для оценки качества работы

Руководителю также доступна страница “Отслеживание времени работы”, где он может изменить график работы отдела, а также получить информацию о времени работы отдельного сотрудника (рисунок 4.29). Для изменения графика работы необходимо выбрать необходимое время и нажать на кнопку “Сохранить”. Для просмотра информации о времени работы отдельного сотрудника необходимо нажать на строку с информацией о сотруднике в таблице, после чего откроется страница о времени работы выбранного сотрудника (рисунок 4.30). На данной странице будет отображено время начала и окончания работы, в зависимости от графика, ячейки будут подсвечены красным цветом при опоздании или раннем уходе, или зелёным, если время соответствует графику работы отдела. Также на странице будет отображена автоматически рассчитанная оценка посещаемости, которую можно будет изменить; для подтверждения оценки нужно будет нажать на кнопку “Установить оценку посещаемости”.

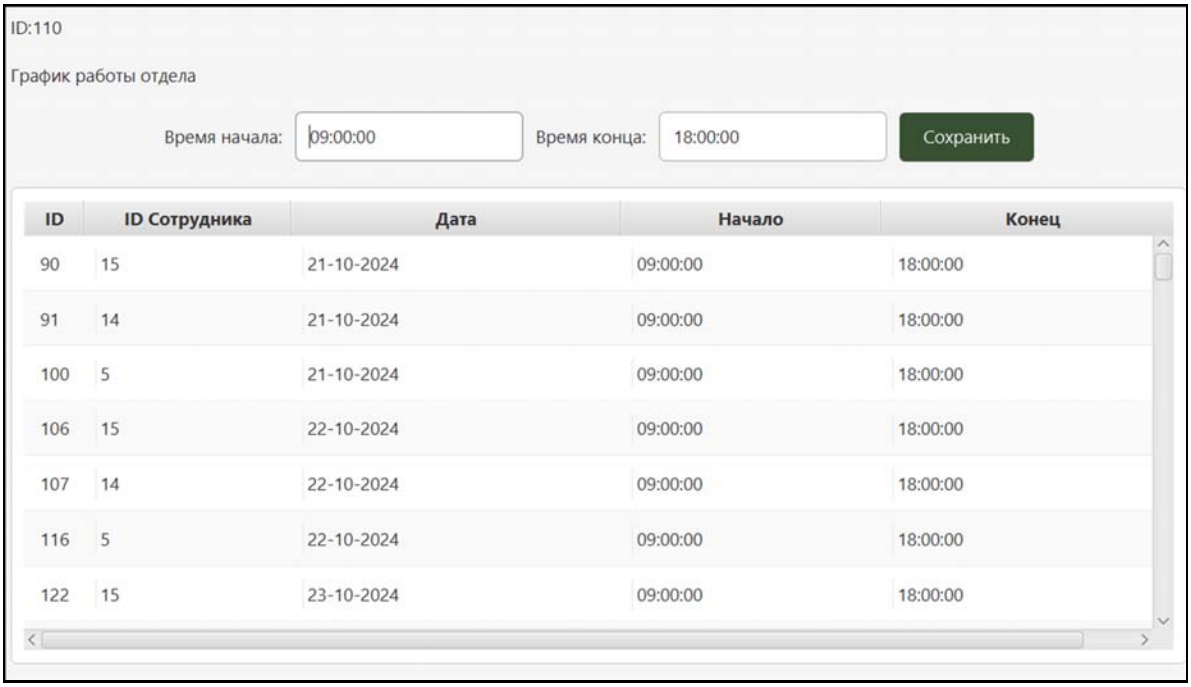


Рисунок 4.29 – Страница “Отслеживание времени работы”

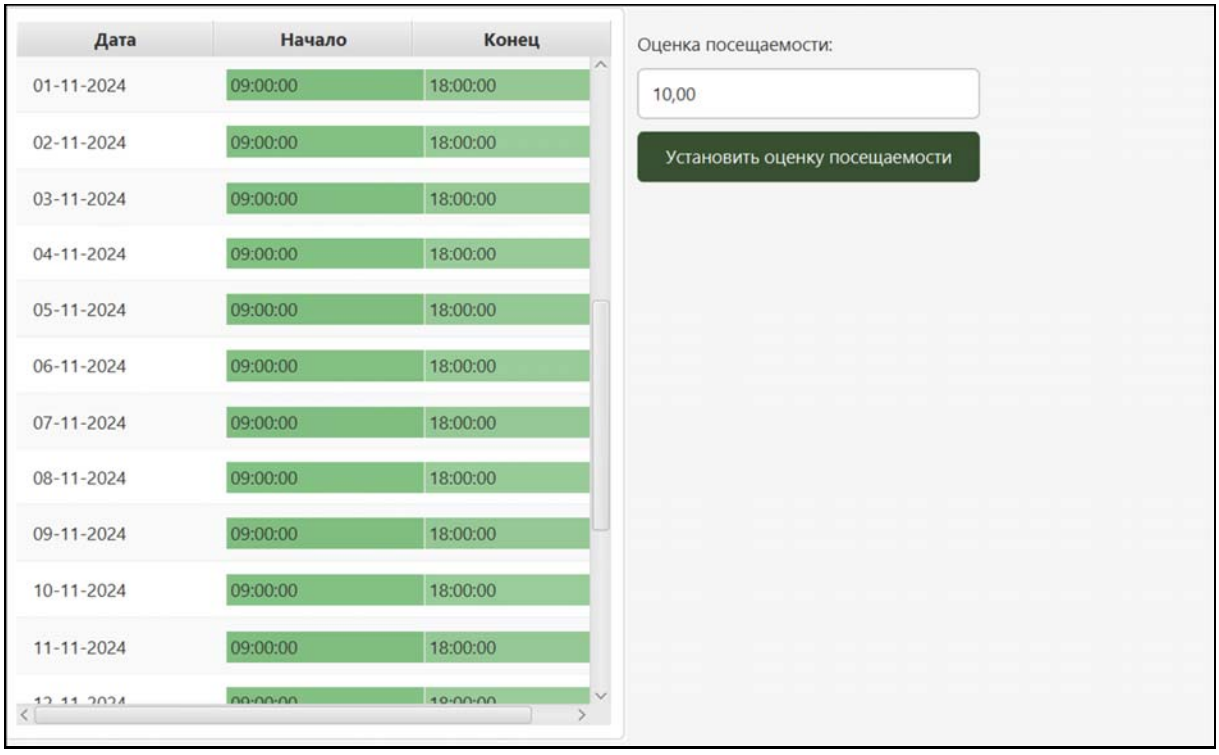


Рисунок 4.30 – Страница с отображением информации о времени работы отдельного сотрудника

Получить доступ к уникальным функциям администратора, сотрудника отдела кадров, сотрудника отдела безопасности и руководителя можно в навигационной панели вверху страницы. Если пользователь является обычным сотрудником, ему не будут доступны никакие уникальные функции, но каждому пользователю, который является сотрудником, будут доступны функции обычного сотрудника. Каждому пользователю доступен трекер времени с таймером на главной странице, который автоматически запускается при запуске программы; для остановки таймера необходимо нажать на кнопку “Стоп”, после чего будут записаны данные о времени работы за день. Также сотрудник может запросить льготу, нажав на кнопку “Запросить льготу”, после чего будет доступно поле с описанием и выпадающий список с типом льготы (рисунок 4.31). Для отображения льгот необходимо нажать на кнопку “Показать льготы” (рисунок 4.32).

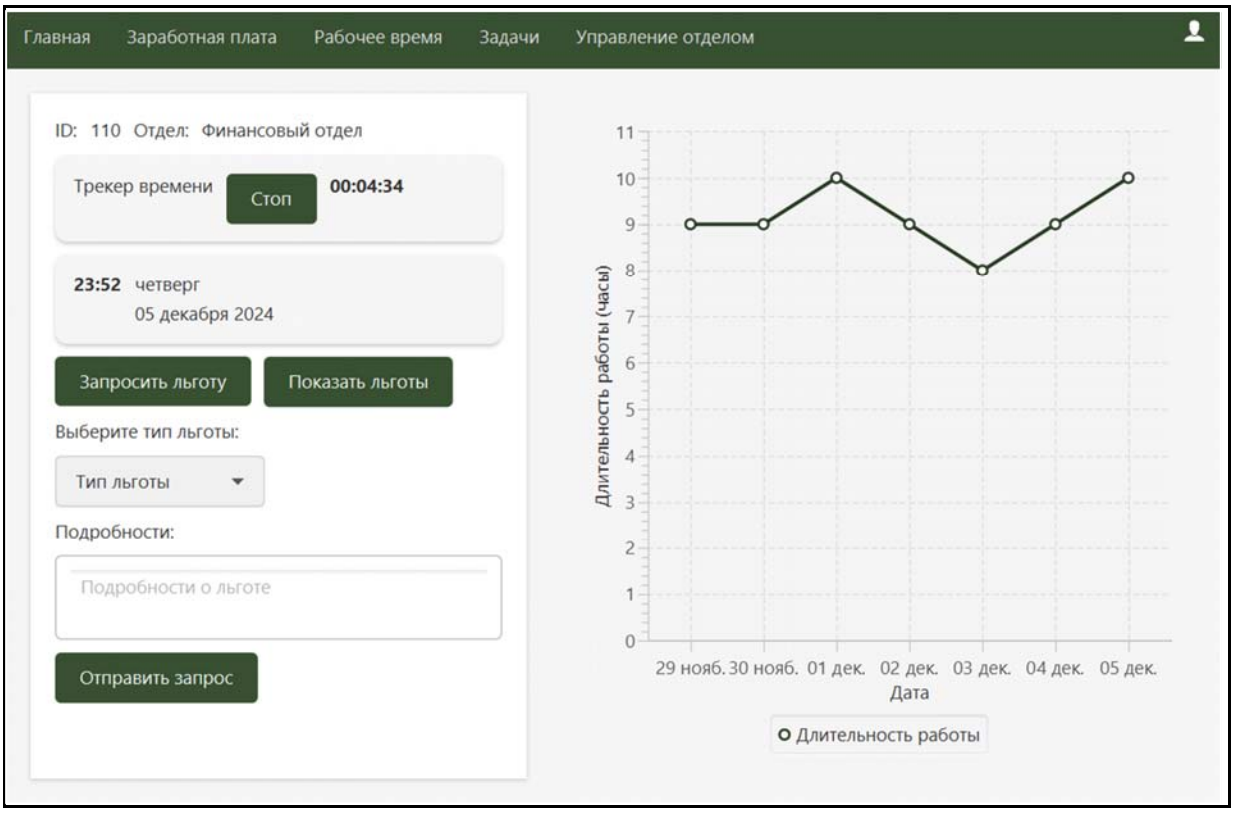


Рисунок 4.31 – Главная страница после нажатия на кнопку “Запросить льготу”

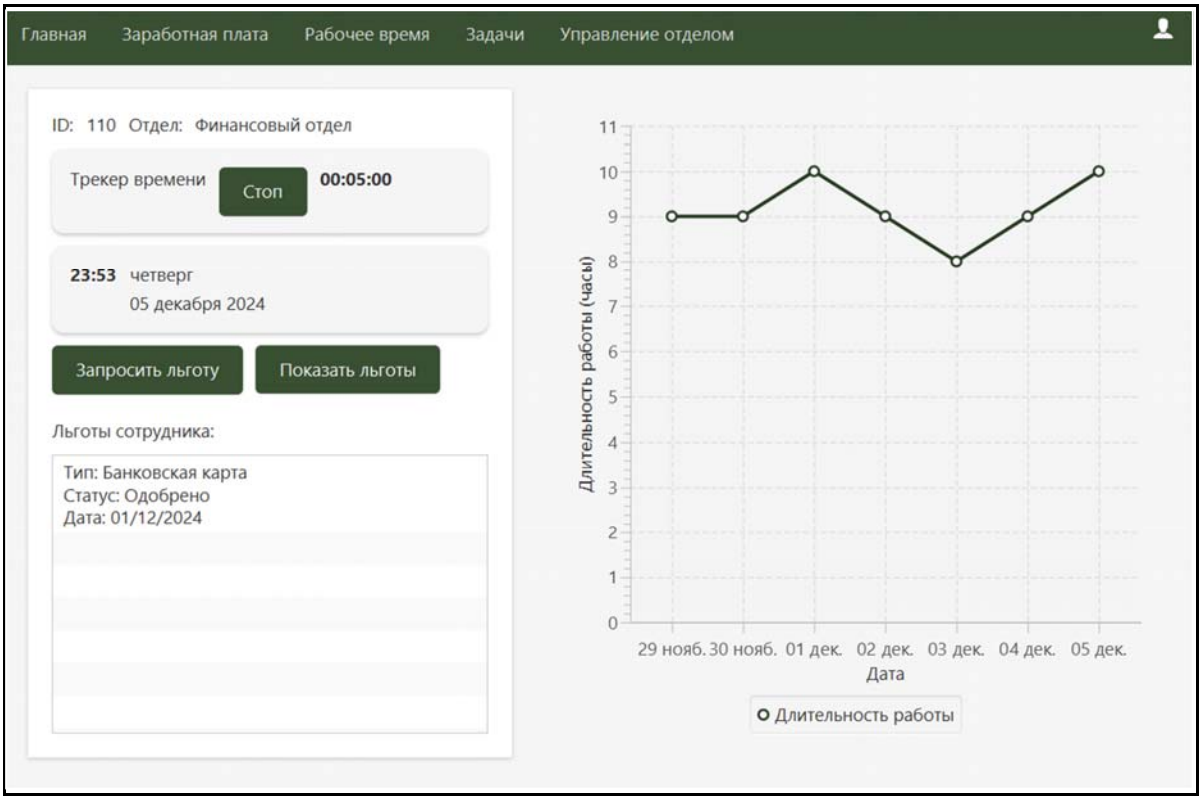


Рисунок 4.32 – Главная страница после нажатия на кнопку “Показать льготы”

В навигационной панели сотрудник может получить доступ к страницам с задачами, заработной платой и рабочим временем. На рисунке 4.33 изображена страница “Рабочее время”, где сотрудник может просматривать информацию о времени работы. На странице “Заработная плата”, которая изображена на рисунке 4.34, можно просмотреть статус заработной платы, а также изменить статус её начисления. На странице “Задачи” сотрудник может просмотреть назначенные задачи и изменить их статус; при нажатии на статус “Завершено” информация о времени окончания и статусе задачи будет сохранена (рисунок 4.35).

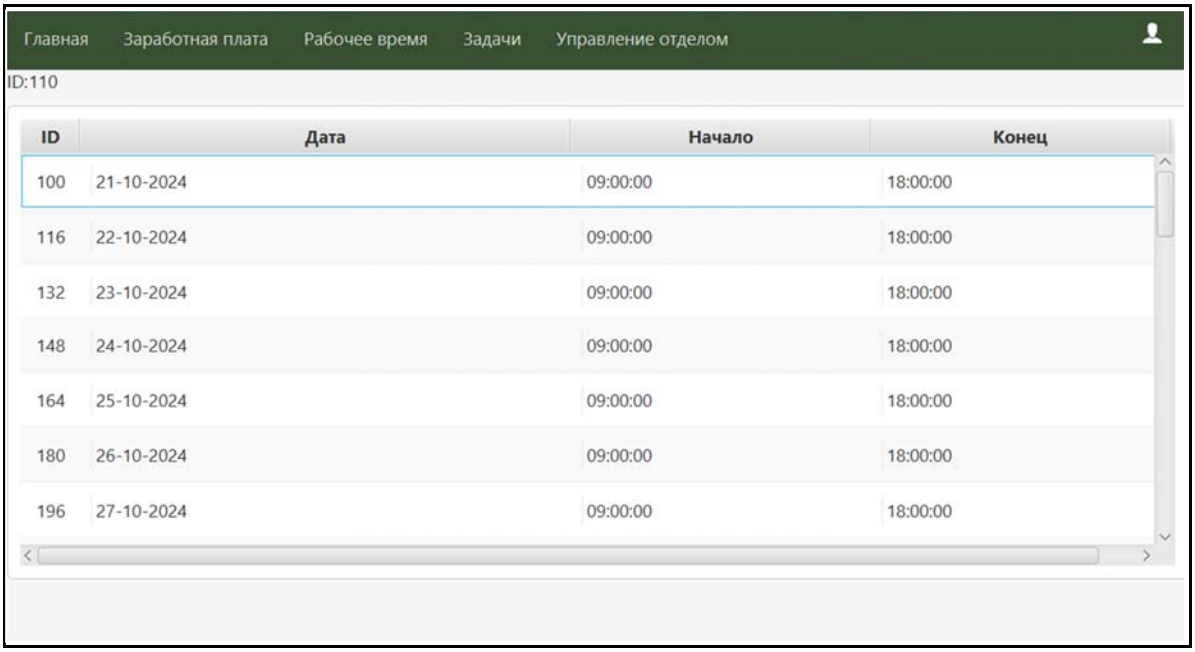


Рисунок 4.33 – Страница “Рабочее время”

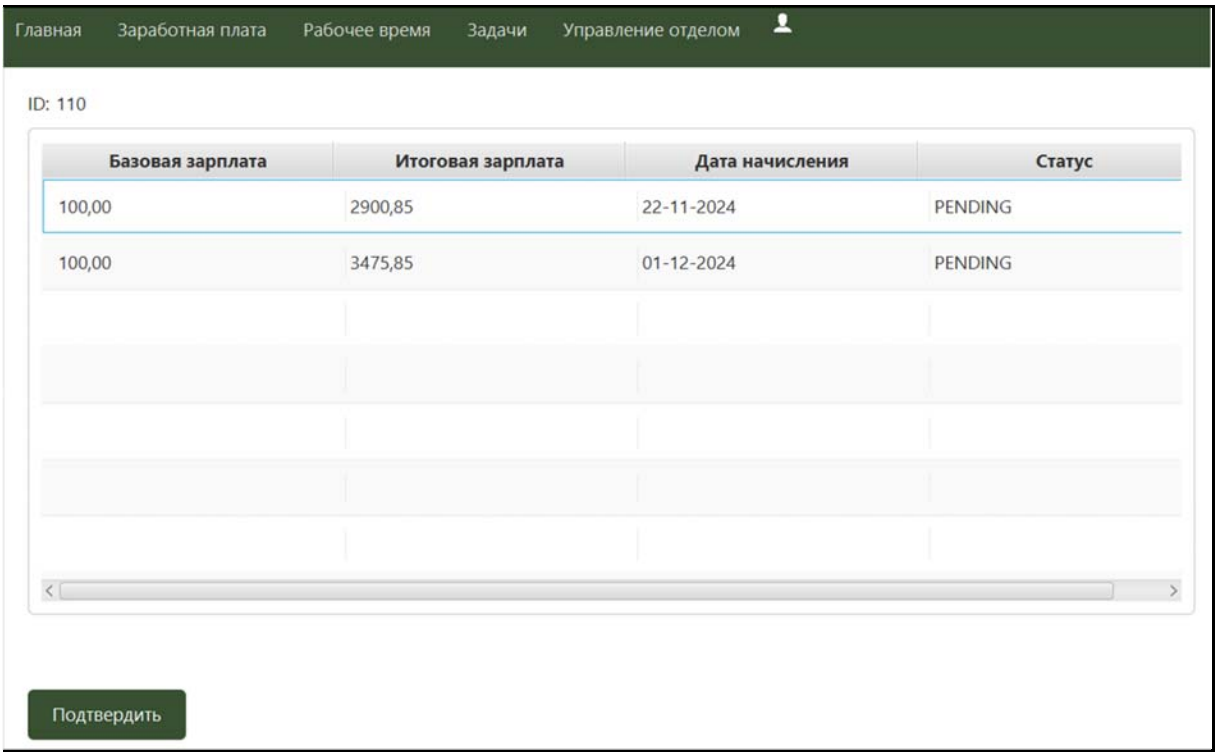


Рисунок 4.34 – Страница “Заработная плата”

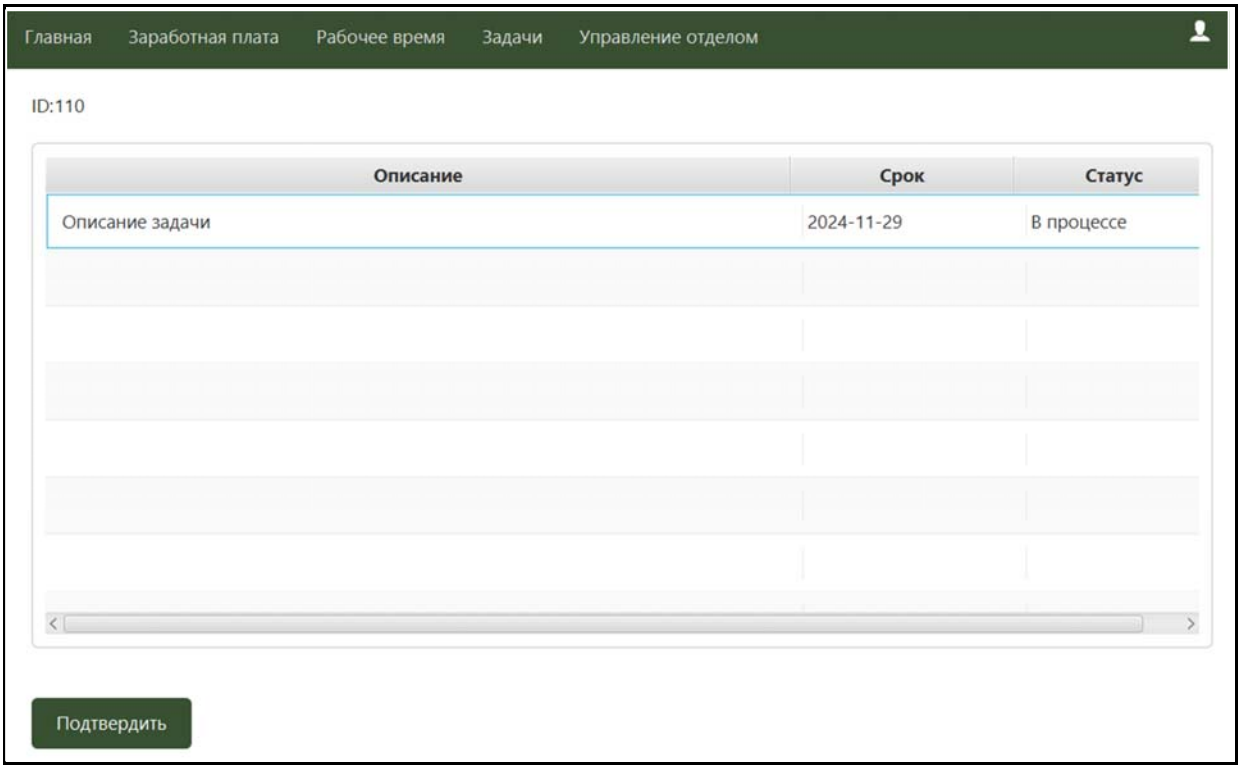


Рисунок 4.35 – Страница “Задачи”

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была создана система автоматизации управления персоналом, направленная на улучшение текущих процессов в банковской сфере. В процессе работы были решены следующие задачи:

– проведён анализ существующих автоматизированных систем управления персоналом, собрана информация о потребностях и особенностях работы банков, разработана функциональная модель и спроектирована информационная структура;

– создан интуитивно понятный интерфейс системы, а также определены оптимальные технологии для её реализации;

– проведено тестирование системы, что подтвердило её надёжность и эффективность в реальных условиях;

– разработано руководство для пользователей, подробно описывающее процесс установки и использования системы.

При разработке были использованы блок-схемы алгоритмов, которые наглядно иллюстрируют функционал системы и её ключевых компонентов. Также были созданы диаграммы бизнес-процессов в форматах IDEF0 и UML.

Работа велась с использованием среды разработки Intellij IDEA и системы управления базами данных MySQL Workbench.

Таким образом, все поставленные цели и задачи успешно выполнены в процессе написания пояснительной записки и создания программного обеспечения. Автоматизированная система управления персоналом позволила значительно повысить качество работы с сотрудниками, улучшить скорость обработки данных и повысить точность расчётов. Руководство банка получило возможность оперативно получать отчёты и анализировать эффективность работы персонала. Разработка данной системы имеет высокий потенциал для увеличения общей эффективности банка и может быть доработана для применения в других областях деятельности. Программный продукт успешно прошёл тестирование и подтвердил свою работоспособность, а в будущем возможно добавление новых функций и улучшений интерфейса для повышения удобства работы с системой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Oracle [Электронный ресурс]. – Режим доступа: https://www.oracle.com/human-capital-management/.
[2] 1С:Зарплата и Управление Персоналом 8 для Беларуси [Электронный ресурс]. – Режим доступа: https://1c.by/v8/generic_products/by_payrollandhr.php.
[3] Тесленко, И. Б., Губернаторов, А. М. Моделирование бизнес-процессов: учебное пособие. – Владимир, 2020.
[4] SAP [Электронный ресурс]. – Режим доступа: https://www.sap.com/central-asia-caucasus/products/hcm/employee-central-hris.html.
[5] Учебное пособие. Диаграмма последовательности [Электронный ресурс]. – Режим доступа: https://creately.com/blog/ru/диаграмма/учебное-пособие-по-последовательной/
[6] Шаллоуэй, А., Тротт, Дж. Паттерны проектирования: Объектно-ориентированный анализ и проектирование. – М.: Питер, 2006. – 46 с.
[7] Буч, Г., Рамбо, Дж., Якобсон, И. Язык UML: Руководство пользователя. – 2-е изд. – М.: ДИАЛЕКТ-МИ, 2005. – 281 с.
[8] Диаграммы развертывания [Электронный ресурс]. – Режим доступа: http://bourabai.ru/dbt/uml/ch30.htm.
[9] Блаха, М., Рамбо, Дж. UML 2.0: Объектно-ориентированное моделирование и разработка. – 2-е изд. – СПб.: Питер, 2007. – 37 с.
[10] Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Вильямс, 2005. – 309 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Отчет о проверке на заимствования в системе «Антиплагиат»

Рисунок А1 – Отчёт о проверке на заимствования

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг кода алгоритмов, реализующих бизнес-логику

Файл CalculateSalaryCommand

```
    private double calculateSalaryForEmployee(
            User employee,
            LocalDate periodStartDate,
            LocalDate periodEndDate,
            double experienceBonusPercent,
            double certificationBonusPercent,
            double qualificationBonusPercent,
            double licenseBonusPercent,
            double sickDayDeductionPercent,
            double vacationDeductionPercent,
            double ratingBonusPercent,
            double attendanceBonusPercent
    ) {
        double baseSalary =
employee.getPersonData().getEmployeeData().getBaseSalary();
        logger.debug("Базовая зарплата для сотрудника {}: {}",
employee.getPersonData().getId(), baseSalary);

        int experience = employee.getPersonData().getExperience();
        int certificationsCount =
countCertifications(employee.getPersonData().getId());
        int qualificationsCount =
countQualifications(employee.getPersonData().getId());
        int licensesCount =
countLicenses(employee.getPersonData().getId());

        int sickDaysCount =
countSickLeaves(employee.getPersonData().getId(), periodStartDate,
periodEndDate);
        logger.debug("Количество больничных дней для сотрудника {}: {}",
employee.getPersonData().getId(), sickDaysCount);

        int vacationDaysCount =
countVacations(employee.getPersonData().getId(), periodStartDate,
periodEndDate);
        logger.debug("Количество отпускных дней для сотрудника {}: {}",
employee.getPersonData().getId(), vacationDaysCount);

        int workedHours =
calculateWorkedHours(employee.getPersonData().getEmployeeData(),
periodStartDate, periodEndDate);
        logger.debug("Отработанные часы для сотрудника {}: {}",
employee.getPersonData().getId(), workedHours);

        double totalSalary = baseSalary;
        totalSalary += experience * experienceBonusPercent;
```

```
        totalSalary += certificationsCount * certificationBonusPercent;
        totalSalary += qualificationsCount * qualificationBonusPercent;
        totalSalary += licensesCount * licenseBonusPercent;

        totalSalary -= sickDaysCount * (sickDayDeductionPercent *
baseSalary);
        totalSalary -= vacationDaysCount * (vacationDeductionPercent *
baseSalary);

        int workingHoursPerDay = calculateWorkingHoursPerDay(employee);
        totalSalary += workedHours * (baseSalary / workingHoursPerDay);

        double attendanceScore =
employee.getPersonData().getEmployeeData().getAttendanceScore();
        totalSalary += attendanceScore * attendanceBonusPercent;

        double ratingScore =
employee.getPersonData().getEmployeeData().getPerformanceRating();
        totalSalary += ratingScore * ratingBonusPercent;

        BigDecimal totalSalaryRounded =
BigDecimal.valueOf(totalSalary).setScale(2, RoundingMode.HALF_UP);
        totalSalary = totalSalaryRounded.doubleValue();

        logger.debug("Итоговая зарплата для сотрудника {}: {}",
employee.getPersonData().getId(), totalSalary);
        return totalSalary;
    }

    @Override
    public void execute(PrintWriter out, Request request) {
        try {
            JsonObject requestMessage =
gson.fromJson(request.getRequestMessage(), JsonObject.class);

            LocalDate payDate =
parseDate(requestMessage.get("payDate").getAsString());
            LocalDate periodStartDate =
parseDate(requestMessage.get("periodStartDate").getAsString());
            LocalDate periodEndDate =
parseDate(requestMessage.get("periodEndDate").getAsString());

            logger.info("Начало расчета зарплаты для периода: {} - {}",
periodStartDate, periodEndDate);

            double experienceBonusPercent =
requestMessage.get("experienceBonusPercent").getAsDouble();
            double certificationBonusPercent =
requestMessage.get("certificationBonusPercent").getAsDouble();
            double qualificationBonusPercent =
requestMessage.get("qualificationBonusPercent").getAsDouble();
            double licenseBonusPercent =
requestMessage.get("licenseBonusPercent").getAsDouble();
```

```
            double                  sickDayDeductionPercent                 =
requestMessage.get("sickDayDeductionPercent").getAsDouble();
            double                vacationDeductionPercent                  =
requestMessage.get("vacationDeductionPercent").getAsDouble();
            double                    ratingBonusPercent                    =
requestMessage.get("ratingBonusPercent").getAsDouble();
            double                  attendanceBonusPercent                  =
requestMessage.get("attendanceBonusPercent").getAsDouble();

            List<User> employees = userService.findAllEntities();
            List<Payroll> payrollList = new ArrayList<>();

            PayrollXmlService payrollXmlService = new PayrollXmlService();

            for (User employee : employees) {
                if   (!"CANDIDATE".equals(employee.getRole_id().getName()))
{
                    logger.info("Расчет  зарплаты  для  сотрудника:  {}",
employee.getPersonData().getId());

                    double                  totalSalary                     =
calculateSalaryForEmployee(employee, periodStartDate, periodEndDate,
                            experienceBonusPercent,
certificationBonusPercent, qualificationBonusPercent,
                            licenseBonusPercent,  sickDayDeductionPercent,
vacationDeductionPercent,
                            ratingBonusPercent, attendanceBonusPercent);

                    Payroll payroll = new Payroll();
                    payroll.setEmployeeId(employee.getPersonData().getEmpl
oyeeData());
                    payroll.setTotalSalary(totalSalary);
                    payroll.setPayDate(java.util.Date.from(payDate.atStart
OfDay(ZoneId.systemDefault()).toInstant()));
                    payroll.setStatus(Payroll.PayrollStatus.PENDING);
                    payrollList.add(payroll);

                    logger.info("Итоговая зарплата для сотрудника {}: {}",
employee.getPersonData().getId(), totalSalary);
                }
            }

            List<PersonData> personDataList = employees.stream()
                    .map(User::getPersonData)
                    .collect(Collectors.toList());

            payrollXmlService.savePayrollDataToXML(payrollList, employees,
personDataList,
                    experienceBonusPercent, certificationBonusPercent,
                    qualificationBonusPercent, licenseBonusPercent,
                    sickDayDeductionPercent, vacationDeductionPercent,
                    ratingBonusPercent, attendanceBonusPercent);
```

```
            Response response = new Response();
            response.setSuccess(true);
            response.setResponseMessage(gson.toJson(payrollList));
            sendResponse(out, response);

            logger.info("Расчет зарплат завершен успешно. Всего обработано
записей: {}", payrollList.size());
        } catch (Exception e) {
            logger.error("Ошибка при обработке запроса: {}",
e.getMessage(), e);
            sendErrorResponse(out, "Ошибка при обработке запроса: " +
e.getMessage());
        }
    }
```

Файл PayrollXMLService

```
    @XmlRootElement(name = "payrollData")
    public class PayrollXmlService {
       private List<PayrollXml> payrolls;
       private List<EmployeeXml> employees;
       private double experienceBonusPercent;
       private double certificationBonusPercent;
       private double qualificationBonusPercent;
       private double licenseBonusPercent;
       private double sickDayDeductionPercent;
       private double vacationDeductionPercent;
       private double hourlyRatePercent;
       private double attendanceBonusPercent;

       public void savePayrollDataToXML(List<Payroll> payrolls, List<User>
users,
                                        List<PersonData> personDataList,
                                        double    experienceBonusPercent,
double certificationBonusPercent,
                                        double qualificationBonusPercent,
double licenseBonusPercent,
                                        double   sickDayDeductionPercent,
double vacationDeductionPercent,
                                        double hourlyRatePercent, double
attendanceBonusPercent) {
           try {
               JAXBContext                    context                    =
JAXBContext.newInstance(PayrollXmlService.class);
               Marshaller marshaller = context.createMarshaller();
               marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);

               List<PayrollXml> payrollXmlList = payrolls.stream()
                       .map(payroll -> {
                           PayrollXml payrollXml = new PayrollXml();
                           payrollXml.setId(payroll.getId());
```

```
                            payrollXml.setTotalSalary(payroll.getTotalSala
ry());
                            payrollXml.setPayDate(payroll.getPayDate());

                            EmployeeXml employeeXml = new EmployeeXml();
                            employeeXml.setId(payroll.getEmployeeId().getI
d());
                            employeeXml.setPosition(payroll.getEmployeeId(
).getPosition());
                            employeeXml.setBaseSalary(payroll.getEmployeeI
d().getBaseSalary());

                            PersonData personData = personDataList.stream()
                                    .filter(pd                              ->
pd.getEmployeeData().getId() == payroll.getEmployeeId().getId())
                                    .findFirst()
                                    .orElse(null);

                            if (personData != null) {
                                employeeXml.setFullName(personData.getFull
Name());
                                employeeXml.setExperience(personData.getEx
perience());
                            } else {
                                System.out.println("PersonData  not  found
for Employee ID: " + payroll.getEmployeeId().getId());
                            }

                            payrollXml.setEmployee(employeeXml);
                            return payrollXml;
                        })
                        .collect(Collectors.toList());

                this.payrolls = payrollXmlList;
                this.experienceBonusPercent = experienceBonusPercent;
                this.certificationBonusPercent                          =
certificationBonusPercent;
                this.qualificationBonusPercent                          =
qualificationBonusPercent;
                this.licenseBonusPercent = licenseBonusPercent;
                this.sickDayDeductionPercent = sickDayDeductionPercent;
                this.vacationDeductionPercent = vacationDeductionPercent;
                this.hourlyRatePercent = hourlyRatePercent;
                this.attendanceBonusPercent = attendanceBonusPercent;

                marshaller.marshal(this,                                new
File("C:\\Users\\BestH\\OneDrive\\Рабочий
стол\\CourseWorkServer\\src\\main\\java\\Utility\\Xml\\payrollData.xml"))
;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

```
        public PayrollXmlService loadPayrollDataFromXML(String filePath) {
            try {
                JAXBContext                    context                    =
JAXBContext.newInstance(PayrollXmlService.class);
                Unmarshaller unmarshaller = context.createUnmarshaller();
                return    (PayrollXmlService)    unmarshaller.unmarshal(new
File(filePath));
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }

        @XmlElement(name = "payroll")
        public List<PayrollXml> getPayrolls() {
            return payrolls;
        }

        public void setPayrolls(List<PayrollXml> payrolls) {
            this.payrolls = payrolls;
        }

        @XmlElement(name = "employee")
        public List<EmployeeXml> getEmployees() {
            return employees;
        }

        public void setEmployees(List<EmployeeXml> employees) {
            this.employees = employees;
        }

        @XmlElement
        public double getExperienceBonusPercent() {
            return experienceBonusPercent;
        }

        public            void            setExperienceBonusPercent(double
experienceBonusPercent) {
            this.experienceBonusPercent = experienceBonusPercent;
        }

        @XmlElement
        public double getCertificationBonusPercent() {
            return certificationBonusPercent;
        }

        public           void           setCertificationBonusPercent(double
certificationBonusPercent) {
            this.certificationBonusPercent = certificationBonusPercent;
        }

        @XmlElement
```

```
        public double getQualificationBonusPercent() {
            return qualificationBonusPercent;
        }

        public          void          setQualificationBonusPercent(double
qualificationBonusPercent) {
            this.qualificationBonusPercent = qualificationBonusPercent;
        }

        @XmlElement
        public double getLicenseBonusPercent() {
            return licenseBonusPercent;
        }

        public void setLicenseBonusPercent(double licenseBonusPercent) {
            this.licenseBonusPercent = licenseBonusPercent;
        }

        @XmlElement
        public double getSickDayDeductionPercent() {
            return sickDayDeductionPercent;
        }

        public           void           setSickDayDeductionPercent(double
sickDayDeductionPercent) {
            this.sickDayDeductionPercent = sickDayDeductionPercent;
        }

        @XmlElement
        public double getVacationDeductionPercent() {
            return vacationDeductionPercent;
        }

        public          void          setVacationDeductionPercent(double
vacationDeductionPercent) {
            this.vacationDeductionPercent = vacationDeductionPercent;
        }


        @XmlElement
        public double getAttendanceBonusPercent() {
            return attendanceBonusPercent;
        }

        public           void           setAttendanceBonusPercent(double
attendanceBonusPercent) {
            this.attendanceBonusPercent = attendanceBonusPercent;
        }
    }
```

ПРИЛОЖЕНИЕ В
(обязательное)
Листинг скрипта генерации базы данных

```
CREATE DATABASE  IF NOT EXISTS `personnel_management`
USE `personnel_management`;
-- MySQL dump 10.13  Distrib 8.0.36, for Win64 (x86_64)
--
-- Host: localhost    Database: personnel_management
-- ------------------------------------------------------
-- Server version     8.0.36


DROP TABLE IF EXISTS `certificates`;
CREATE TABLE `certificates` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `certificate_name` varchar(150) DEFAULT NULL,
  `issued_by` varchar(100) DEFAULT NULL,
  `issue_date` date DEFAULT NULL,
  `expiry_date` date DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `employee_id_idx` (`employee_id`),
  CONSTRAINT  `employee_id`  FOREIGN  KEY  (`employee_id`)  REFERENCES
`personal_data` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)

DROP TABLE IF EXISTS `department`;
CREATE TABLE `department` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `work_schedule_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `work_schedule_id` (`work_schedule_id`),
  CONSTRAINT  `department_ibfk_1`  FOREIGN  KEY  (`work_schedule_id`)
REFERENCES `work_schedule` (`id`) ON DELETE SET NULL
)


DROP TABLE IF EXISTS `employee_benefits`;
CREATE TABLE `employee_benefits` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `benefit_type` varchar(255) DEFAULT NULL,
  `details` text,
  `status` varchar(50) DEFAULT NULL,
  `created_date` date DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `employee_id` (`employee_id`),
  CONSTRAINT  `employee_benefits_ibfk_1`  FOREIGN  KEY  (`employee_id`)
REFERENCES `employees` (`id`) ON DELETE CASCADE
)
```

```
DROP TABLE IF EXISTS `employees`;
CREATE TABLE `employees` (
  `id` int NOT NULL AUTO_INCREMENT,
  `hire_date` date NOT NULL,
  `position` varchar(100) DEFAULT NULL,
  `termination_date` date DEFAULT NULL,
  `department_id` int DEFAULT NULL,
  `performance_rating` int DEFAULT NULL,
  `attendance_score` decimal(3,2) DEFAULT NULL,
  `base_salary` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `department_id` (`department_id`),
  KEY `personaldatar_id_idx` (`id`),
  CONSTRAINT `employees_ibfk_2` FOREIGN KEY (`department_id`) REFERENCES
`department` (`id`) ON DELETE SET NULL
)


DROP TABLE IF EXISTS `incidents`;
CREATE TABLE `incidents` (
  `id` int NOT NULL AUTO_INCREMENT,
  `description` text,
  `date` date DEFAULT NULL,
  `user_id` int DEFAULT NULL,
  `severity` enum('LOW','MEDIUM','HIGH') DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `incidents_ibfk_1_idx` (`user_id`),
  CONSTRAINT  `incidents_ibfk_1`  FOREIGN  KEY  (`user_id`)  REFERENCES
`employees` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)

DROP TABLE IF EXISTS `interviews`;
CREATE TABLE `interviews` (
  `id` int NOT NULL AUTO_INCREMENT,
  `candidate_id` int DEFAULT NULL,
  `interviewer_id` int DEFAULT NULL,
  `interview_date` datetime DEFAULT NULL,
  `status` enum('SCHEDULED','COMPLETED','CANCELED') DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `candidate_id_idx` (`candidate_id`),
  KEY `interviewer_id_idx` (`interviewer_id`),
  CONSTRAINT  `candidate_id`  FOREIGN  KEY  (`candidate_id`)  REFERENCES
`personal_data` (`user_id`),
  CONSTRAINT `interviewer_id` FOREIGN KEY (`interviewer_id`) REFERENCES
`employees` (`id`)
)
DROP TABLE IF EXISTS `licenses`;
CREATE TABLE `licenses` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `license_name` varchar(100) DEFAULT NULL,
  `issued_by` varchar(100) DEFAULT NULL,
  `issue_date` date DEFAULT NULL,
  `expiry_date` date DEFAULT NULL,
```

```
  PRIMARY KEY (`id`),
  KEY `employee_fk_idx` (`employee_id`),
  CONSTRAINT `employee_fk` FOREIGN KEY (`employee_id`) REFERENCES
`personal_data` (`user_id`)
)


DROP TABLE IF EXISTS `payroll`;
CREATE TABLE `payroll` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `total_salary` decimal(10,2) DEFAULT NULL,
  `pay_date` date DEFAULT NULL,
  `status` varchar(45) DEFAULT NULL,
  `period_start_date` date DEFAULT NULL,
  `period_end_date` date DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `payroll_ibfk_1` (`employee_id`),
  CONSTRAINT `payroll_ibfk_1` FOREIGN KEY (`employee_id`) REFERENCES
`employees` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `personal_data`;
CREATE TABLE `personal_data` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `full_name` varchar(150) DEFAULT NULL,
  `date_of_birth` date DEFAULT NULL,
  `phone` varchar(30) DEFAULT NULL,
  `email` varchar(100) DEFAULT NULL,
  `employee_data_id` int DEFAULT NULL,
  `experience` int DEFAULT NULL,
  `passport_number` varchar(20) DEFAULT NULL,
  `place_of_birth` varchar(100) DEFAULT NULL,
  `address` varchar(255) DEFAULT NULL,
  `marital_status` varchar(50) DEFAULT NULL,
  `family_members` text,
  PRIMARY KEY (`user_id`),
  KEY `employee_id_idx` (`employee_data_id`),
  CONSTRAINT `employee_data_id` FOREIGN KEY (`employee_data_id`)
REFERENCES `employees` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `qualifications`;
CREATE TABLE `qualifications` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `degree` varchar(100) DEFAULT NULL,
  `institution` varchar(100) DEFAULT NULL,
  `graduation_year` year DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `employee_id_idx` (`employee_id`),
```

```
  CONSTRAINT  `employee_fk_1`  FOREIGN  KEY  (`employee_id`)  REFERENCES
`personal_data` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `resumes`;
CREATE TABLE `resumes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `candidate_id` int DEFAULT NULL,
  `summary` text,
  `skills` text,
  `education` text,
  `languages` text,
  `status` enum('APPROVED','WAITING','CANCELED') DEFAULT 'WAITING' COMMENT
'\n        \n        ',
  `background_check_status` tinyint DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `employee_fk_2_idx` (`candidate_id`),
  CONSTRAINT  `employee_fk_2`  FOREIGN  KEY  (`candidate_id`)  REFERENCES
`personal_data` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)


DROP TABLE IF EXISTS `role`;
CREATE TABLE `role` (
  `id` int NOT NULL,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
)


DROP TABLE IF EXISTS `sick_leaves`;
CREATE TABLE `sick_leaves` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `end_date` date DEFAULT NULL,
  `reason` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `sick_leaves_ibfk_1` (`employee_id`),
  CONSTRAINT `sick_leaves_ibfk_1` FOREIGN KEY (`employee_id`) REFERENCES
`employees` (`id`) ON DELETE CASCADE
)
UNLOCK TABLES;


DROP TABLE IF EXISTS `task_type`;
CREATE TABLE `task_type` (
  `id` int NOT NULL AUTO_INCREMENT,
  `certificate` varchar(100) DEFAULT NULL,
  `license` varchar(100) DEFAULT NULL,
  `qualification` varchar(100) DEFAULT NULL,
  `title` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
```

```
)


DROP TABLE IF EXISTS `tasks`;
CREATE TABLE `tasks` (
  `id` int NOT NULL AUTO_INCREMENT,
  `title` int DEFAULT NULL,
  `description` text,
  `status` enum('Не начата','В процессе','Завершена') DEFAULT 'Не начата',
  `end_date` date DEFAULT NULL,
  `due_date` date DEFAULT NULL,
  `employee_id` int DEFAULT NULL,
  `complexity` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `emoloyee_id_idx` (`employee_id`),
  KEY `task_type_idx` (`title`),
  CONSTRAINT `emoloyee_id` FOREIGN KEY (`employee_id`) REFERENCES
`employees` (`id`),
  CONSTRAINT `task_type_id` FOREIGN KEY (`title`) REFERENCES `task_type`
(`id`)
)

DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `role_id` int DEFAULT NULL,
  `personal_data_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `user_id_idx` (`personal_data_id`),
  KEY `role_id_idx` (`role_id`),
  CONSTRAINT `role_id` FOREIGN KEY (`role_id`) REFERENCES `role` (`id`),
  CONSTRAINT `user_id` FOREIGN KEY (`personal_data_id`) REFERENCES
`personal_data` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)
UNLOCK TABLES;


DROP TABLE IF EXISTS `vacations`;
CREATE TABLE `vacations` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `end_date` date DEFAULT NULL,
  `status` enum('PENDING','APPROVED','REJECTED') DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `employee_id_fk_idx` (`employee_id`),
  CONSTRAINT `employee_id_fk` FOREIGN KEY (`employee_id`) REFERENCES
`employees` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)


LOCK TABLES `vacations` WRITE;
```

```
UNLOCK TABLES;

DROP TABLE IF EXISTS `work_schedule`;
CREATE TABLE `work_schedule` (
  `id` int NOT NULL AUTO_INCREMENT,
  `start_time` time DEFAULT NULL,
  `end_time` time DEFAULT NULL,
  PRIMARY KEY (`id`)
)


LOCK TABLES `work_schedule` WRITE;
UNLOCK TABLES;

DROP TABLE IF EXISTS `work_time`;
CREATE TABLE `work_time` (
  `id` int NOT NULL AUTO_INCREMENT,
  `employee_id` int DEFAULT NULL,
  `start_time` time DEFAULT NULL,
  `end_time` time DEFAULT NULL,
  `date` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `work_time_ibfk_1` (`employee_id`),
  CONSTRAINT `work_time_ibfk_1` FOREIGN KEY (`employee_id`) REFERENCES
`employees` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

ПРИЛОЖЕНИЕ Г
(обязательное)
Ведомость документа

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1-40 05 01-02 017 ПЗ	Пояснительная записка	93 с.
	Графические документы	
ГУИР.502528.017 ПД	Чертёж Схема функциональной	Формат А4
	модели блока «Начислить	
	заработную плату»	
ГУИР.502528.017 ПД	Чертёж Схема алгоритма,	Формат А4
	реализующая бизнес-логику	
	программного средства	
ГУИР.502528.017 ПЛ	Плакат UML диаграмма классов	Формат А4
ГУИР.502528.017 ПЛ	Плакат Модели представления	Формат А4
	программного средства	
ГУИР.502528.017 ПЛ	Плакат Скриншоты рабочих	Формат А4
	окон программного средства	

					БГУИР КР 1-40 05 01-02 017 Д1		
Изм.	*Л.*	*№ докум.*	*Подп.*	*Дата*	*Разработка системы управления персоналом в банковской сфере*	*Лист*	*Листов*
Разраб.		*Дубровская А.М.*				*1*	*1*
Пров.		*Сильванович Ю.В.*			*Ведомость курсового проекта*	*Кафедра ЭИ гр. 272303*	
Н.контр.		*Сильванович Ю.В.*					