Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.Ю. Петрович |
|  | \_\_\_\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«**Программная поддержка учета и анализа деятельности персонала на предприятии**»**

БГУИР КП 1-40 05 01-09 051 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 010101  Листванович Александр Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_\_\_\_\_.2022  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

**РЕФЕРАТ**

БГУИР КП 1-40 05 01-10 009 ПЗ

Листванович А.А. Программная поддержка деятельности компании по предоставлению туристических услуг / А. Листванович – Минск: БГУИР, 2022.

Пояснительная записка состоит из 65 страниц, 41 рисунок, 4 приложения.

ПРИЛОЖЕНИЕ, ПРЕДПРИЯТИЕ, ДЕСКТОП ПРИЛОЖЕНИЕ, АДМИНИСТРАТОР, УЧЕТ, АНАЛИЗ, ПОЛЬЗОВАТЕЛЬ, КЛИЕНТ, СЕРВЕР, КОНТРОЛЛЕР, МОДЕЛЬ, ВИД, ИНТЕРФЕЙС, ПАТТЕРН, КОНСТРУКЦИЯ, ШАБЛОН, ПРИЛОЖЕНИЕ.

*Цель проектирования*: разработка десктоп приложения для программной поддержки учета и анализа деятельности персонала на предприятии.

*Методология проведения работы*: в процессе решения поставленных задач были использованы принципы и методы паттерна MVC (Модель-Представление-Контроллер), методы защиты приложения от посторонних пользователей, были использованы конфигурации платформы для авторизации и работы с данными.

*Результаты работы*: выполнена обработка процедуры добавления персонала предприятии, обработано назначение дисциплинарных взысканий и премирование сотрудников. Реализован функционал по расчету заработной платы персонала, с учетом почасовой ставки, премий, взысканий и режима работы. Уделено внимание технико-экономическим вопросам данного проекта, который представлен в виде статистики отделов предприятия по разным критериям.

*Область применения результатов*: могут быть использованы при проектировании десктоп приложения с удобным интерфейсом для программной поддержки предприятия.

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc121088445)

[1 Анализ и моделирование предметной области программного средства 7](#_Toc121088446)

[1.1 Описание предметной области 7](#_Toc121088447)

[1.2 Разработка функциональной модели предметной области 7](#_Toc121088448)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 11](#_Toc121088449)

[1.4 Разработка информационной модели предметной области 12](#_Toc121088450)

[1.5 UML-модели представления программного средства и их описание 14](#_Toc121088451)

[2 Проектирование и конструирование программного средства 16](#_Toc121088452)

[2.1 Постановка задачи 16](#_Toc121088453)

[2.2 Архитектурные решения 17](#_Toc121088454)

[2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства 18](#_Toc121088455)

[2.4 Проектирование пользовательского интерфейса 19](#_Toc121088456)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 20](#_Toc121088457)

[3 Тестирование и проверка работоспособности программного средства 21](#_Toc121088458)

[4 Инструкция по развертыванию приложения и сквозной тестовый пример, начиная от авторизации, демонстрируя реализацию всех вариантов использования 22](#_Toc121088459)

[Заключение 44](#_Toc121088460)

[Список использованных источников 46](#_Toc121088461)

[Приложение А](#_Toc121088462) [(обязательное)](#_Toc121088463) [UML Диаграммы классов 47](#_Toc121088464)

[Приложение Б](#_Toc121088465) [(обязательное)](#_Toc121088466) [Функциональная модель процессов функционирования системы управления туристической фирмы 49](#_Toc121088467)

[Приложение В](#_Toc121088468) [(обязательное)](#_Toc121088469) [Листинг кода 50](#_Toc121088470)

[Приложение Г](#_Toc121088471) [(обязательное)](#_Toc121088472) [Скрипт генерации базы данных 64](#_Toc121088473)

# ВВЕДЕНИЕ

В настоящее время во всех отраслях человеческой деятельности огромное внимание уделяется использованию информационных технологий для наиболее рационального и эффективного использования ресурсов, имеющихся в распоряжении человека.

Среди многообразия поисков путей развития рынка, средств производства, новых направлений деятельности коммерческо-посреднических организаций и предприятий вызывают значительный интерес научные исследования и практические новации. В течение последних лет бурно развиваются основанные на информатике новые технологии.

Актуальность темы заключается в том, что информационные системы (ИС) занимают в этих технологиях центральное положение.

В современных условиях эффективное управление представляет собой ценный ресурс организации, наряду с финансовыми, материальными, человеческими и другими ресурсами. Следовательно, повышение эффективности становится одним из направлений совершенствования деятельности в целом. Наиболее очевидным способом повышения эффективности протекания трудового процесса является его автоматизация. Бурное развитие информационных компьютерных технологий, совершенствование технической платформы и появление принципиально новых классов программных продуктов привело к изменению подходов к автоматизации управления предприятиями, автоматизации документооборота, автоматизации различных процессов предприятий.

Начисление амортизации основных средств обычно не представляет никаких методических трудностей, однако на практике обычно представляет собой однообразную, повторяющуюся из месяца в месяц операцию. В отдельных случаях управляющему предприятием могут потребоваться данные по сумме, которую он должен потратить на содержание сотрудников на выплату им заработных плат. Кроме того, необходимо контролировать сумму начисленной заработной платы, чтобы не допустить его излишнее начисление. Так, основным результатом автоматизации этого участка учета должно стать освобождение бухгалтера от однообразной работы. Программа учета и анализа персонала на предприятии должна также позволять формировать различные отчеты и вести аналитический учет в разрезе отдельных объектов основных средств.

Каждое производственная организация имеет в своем распоряжении основные и оборотные средства. Поэтому основной целью является создание базы данных, которая позволит структурировать те огромные объемы

информации, которые накапливаются на предприятиях, и тем самым существенно облегчить работу и доступ к данным практически всем звеньям, участвующим в управлении основными средствами предприятия.

Таким образом, целью данного курсового проекта является сокращение временных затрат управляющих предприятием лиц за счёт автоматизации учета и анализа необходимой информации о персонале: расчета заработных плат, учета дисциплинарных взысканий и премиальных поощрений сотрудникам представления расчётов в виде диаграмм.

Для достижения поставленной цели требуется решить следующие задачи:

* проанализировать заданную предметную область;
* рассмотреть физическую и логическую модель представления данных;
* создать базу данных;
* реализовать серверную часть, имеющую доступ к базе данных и выполняющую ряд необходимых действий, позволяющих достигнуть цель курсового проекта;
* реализовать клиентскую часть с дружественным интерфейсом, обеспечивающий максимально удобное взаимодействие с программой или вычислительной системой;
* реализовать в виде круговых графиков информацию по каждому отделу.

# 1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

## Описание предметной области

В данной курсовой работе необходимо разработать информационную систему для учета и анализа персонала на предприятии. Главной задачей при разработке данной информационной системы является быстрота нахождения информации, просмотра списка сотрудников, учета их деятельности путем учета отработанных каждым сотрудником часов и учета их дисциплинарных взысканий и поощрений." Программная поддержка учета и анализа деятельности персонала на предприятии ". ИС должна быть проста в использовании и предельно понятна для сотрудников. Клиент не может вносить изменения в структуру таблиц и схему данных.

«Программная поддержка учета и анализа деятельности персонала на предприятии» должна обеспечивать выполнение следующих основных функций:

* ведение списка сотрудников на предприятии, учета их деятельности как каждого по отдельности, так и по определенным отделам предприятия;
* вывод информации о сотрудниках;
* вывод информации о деятельности сотрудников, наличие дисциплинарных взысканий, премиальных поощрений, количестве отработанных часов каждым сотрудником (или отделом в целом);
* расчет заработной платы для каждого сотрудника в отдельности в зависимости от наличия у сотрудника дисциплинарного взыскания или поощрения и количестве отработанных сотрудником часов;
* вывод информации о заработной плате для сотрудников;
* вывод информации по отделам о количестве дисциплинарных взысканий количестве сотрудников и количестве отработанных часов в круговых диаграммах.

## 1.2 Разработка функциональной модели предметной области

Анализ предметной области является одним из важнейших этапов проектирования информационной системы. На данном этапе необходимо понять и описать бизнес-логику предметной области. Известно, что исправление ошибок, допущенных на данной стадии, обходится примерно в десять раз дороже, чем на следующих, откуда следует, что начальные стадии проекта являются одними из наиболее критических и требуют применения эффективных средства автоматизации.

Для графического описания модели системы предприятии был использован стандарт **IDEF0** и инструментальная среда **BpWin**.

**IDEF0** — это методология графического описания систем и процессов деятельности организации как множества взаимосвязанных функций. Она позволяет исследовать функции организации, не связывая их с объектами, обеспечивающими их реализацию [1].

В стандарте **IDEF0** посредством входа показывают объекты — информационные и материальные потоки, которые преобразуются в бизнес- процессе. С помощью управления показываются объекты — материальные и информационные потоки, которые не преобразуются в процессе, по нужны для его выполнения. Используя механизмы IDEF0 можно отображать инструменты и ресурсы, с помощью которых бизнес-процесс реализуется (например, технические средства, люди, информационные системы и т.д.). Выход бизнес-процесса, описанного в стандарте IDEF0, полностью соответствует по смыслу выходу процесса, описанному с помощью DFD-схемы.

Детальное рассмотрение процесса учета и анализа деятельности персонала на предприятии реализовано с помощью нотации IDEF0, которая широко применяется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Нотация IDEF0 является одной из самых популярных нотаций моделирования бизнес-процессов. На первом уровне декомпозиции используется «контекстная диаграмма». Стрелки на этой диаграмме отображают связи объекта моделирования с окружающей средой.

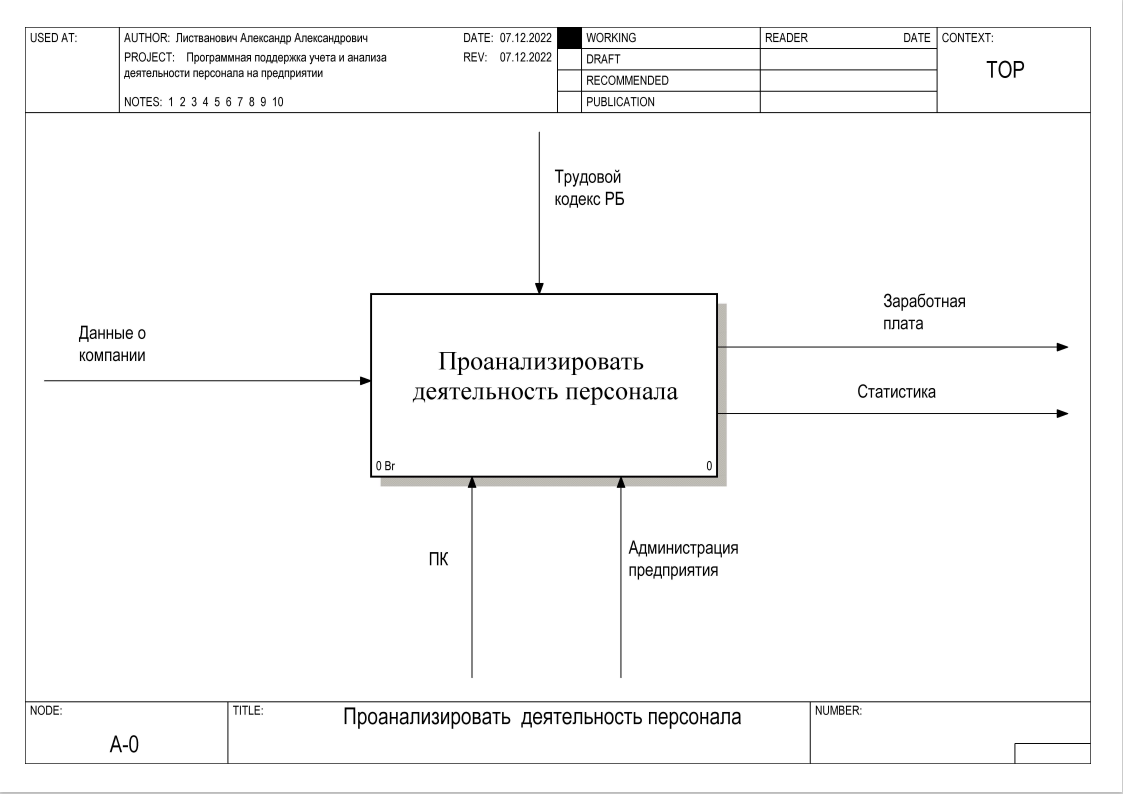


Рисунок 1.1 – Контекстная диаграмма

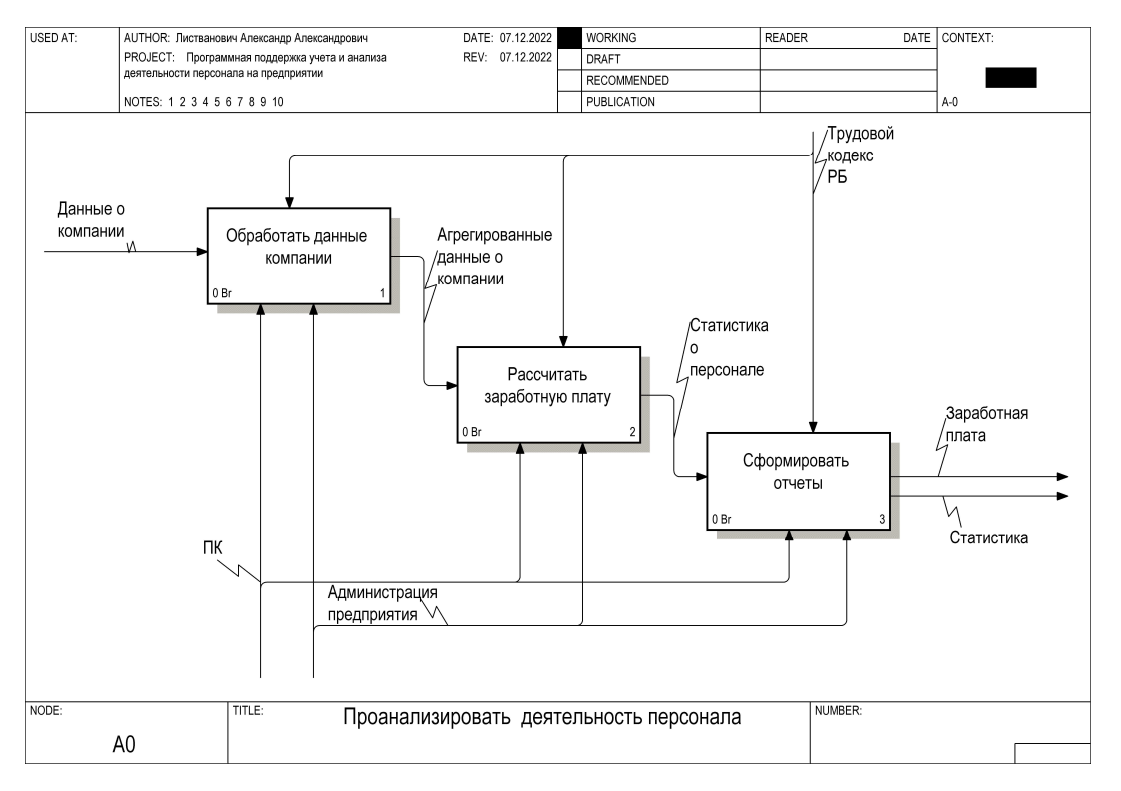


Рисунок 1.2 – Первый уровень декомпозиции

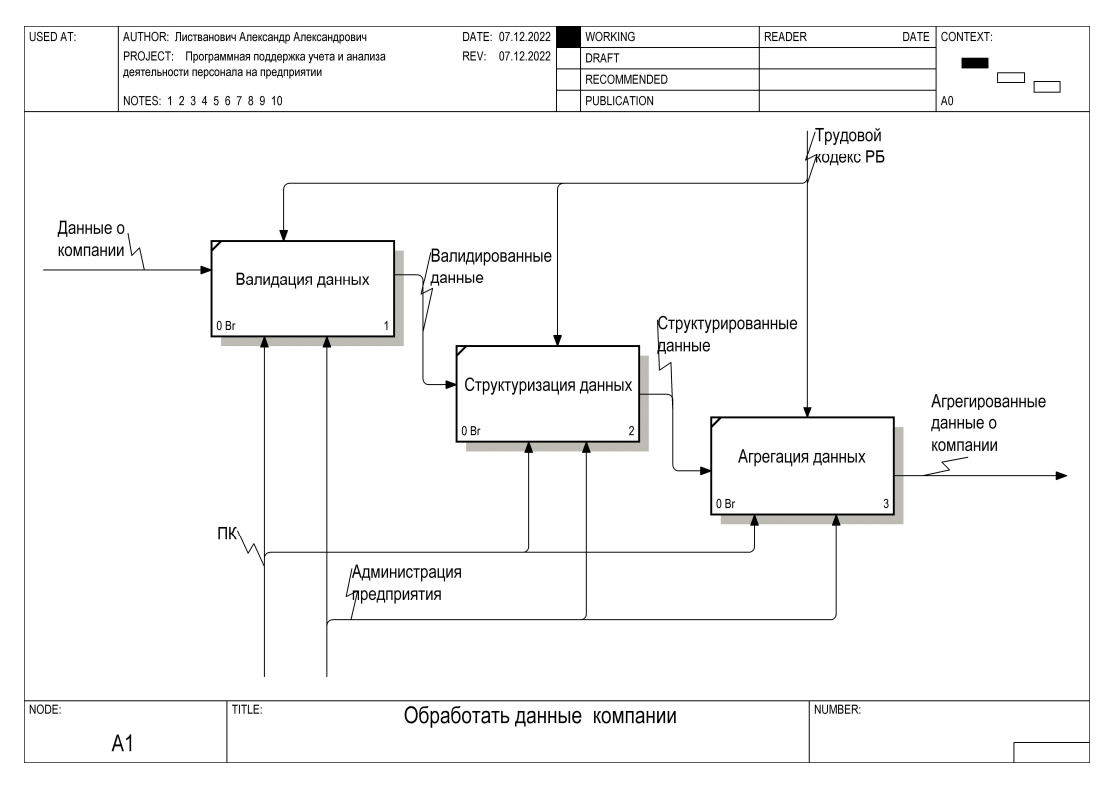


Рисунок 1.3 – Декомпозиция функционального блока «Обработать данные компании»

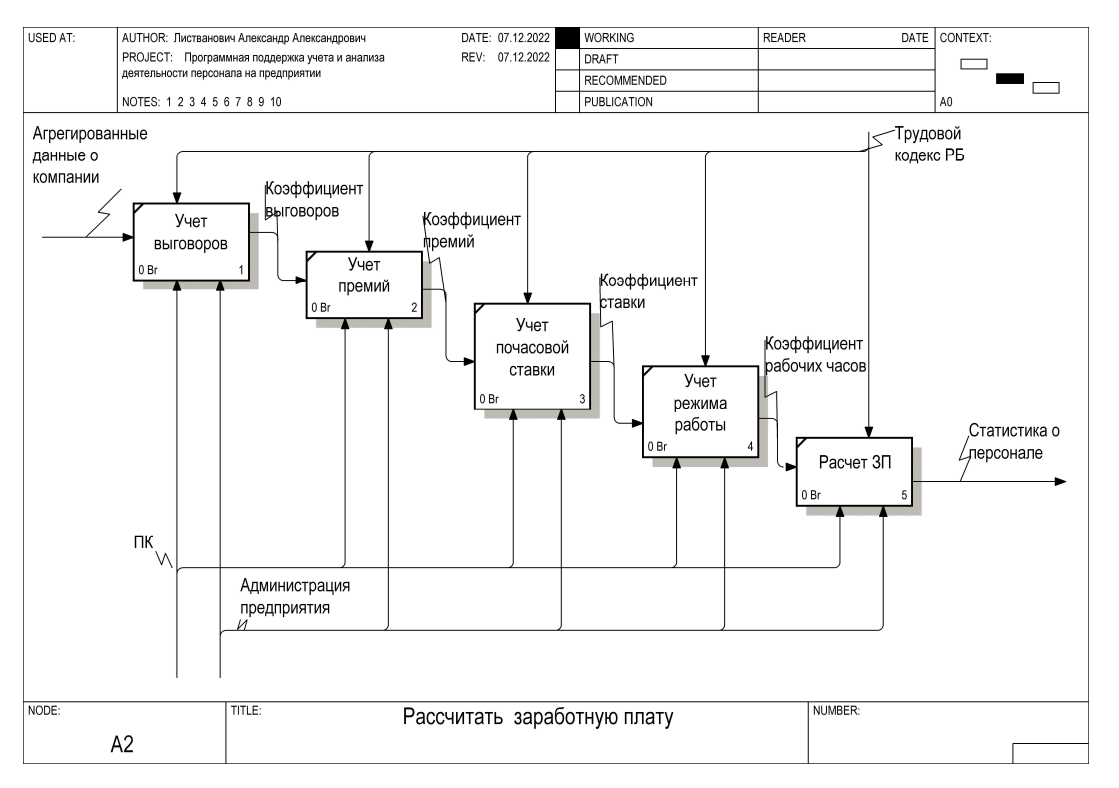


Рисунок 1.4 – Декомпозиция функционального блока «Рассчитать заработную плату»

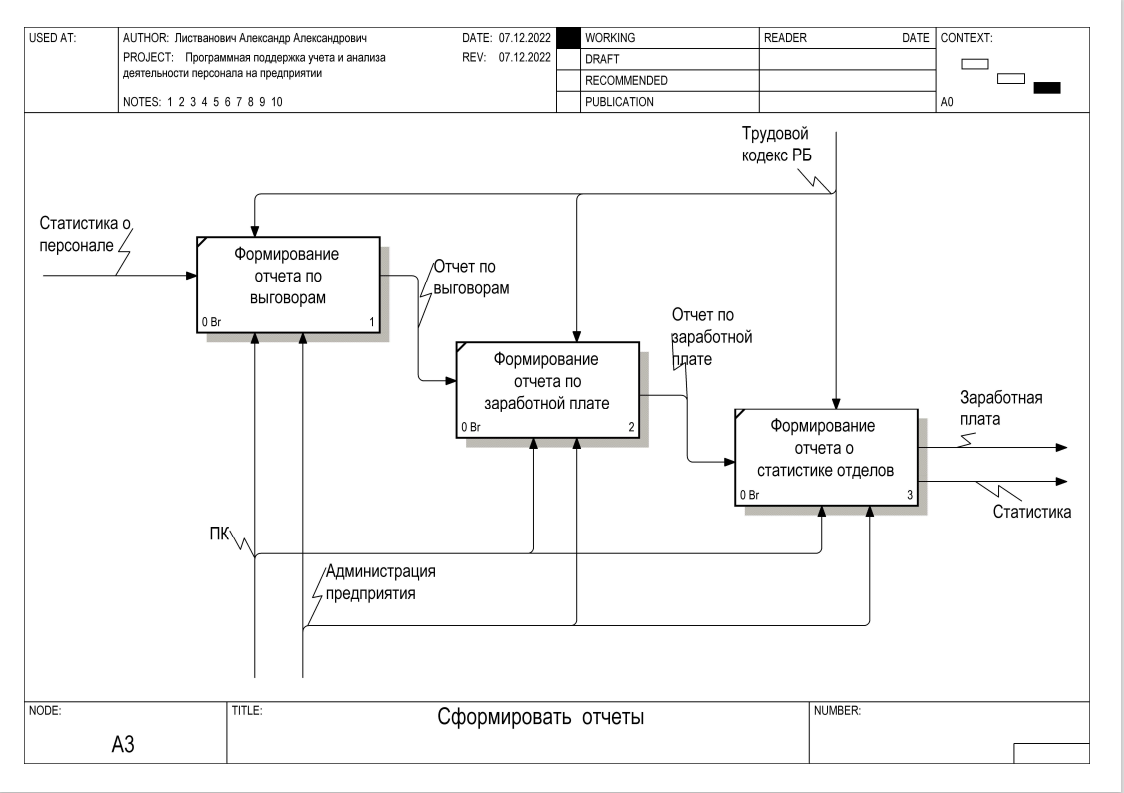


Рисунок 1.5 – Декомпозиция функционального блока «Сформировать отчеты»

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

Задача данного курсового проекта – это создание функционирующего приложения в архитектуре клиент-сервер с организацией взаимодействия с базой данных на объектно-ориентированном языке Java. Серверное приложение может быть реализовано в виде консольного приложения или GUI-приложения. Клиентское приложение: оконное приложение.

Рассмотрим некоторые функции, реализующие бизнес-логику:

* регистрация пользователей;

При регистрации пользователю предоставляется окно регистрации, где необходимо создать новую учетную запись. Данные отправляются на сервер и сохраняются в БД.

* авторизация;

Пользователь вводит свой логин и пароль. Если введённый логин есть в базе данных, то далее пароль хешируется и проверяется, совпадает ли хеш пароля, введённого пользователем с хешем в базе данных. Если логин в БД не найдена или введён не верный пароль, то пользователю будет выведено сообщение об ошибке. Иначе выведется сообщение об успешном входе и в зависимости от роли пользователя, у него будут определённые привилегии.

* просмотр информации;

Клинт посылает сообщения серверу о получения данные о заработной плате, после чего на сервере формируется SQL-запрос и нужная информация отправляется на клиент для отображения пользователю.

* просмотр операций;

Формируется SQL-запрос, и выводится информация, о об операциях.

* визуализация информации посредством круговых диаграмм;

С сервера заращивается статистика об операциях и клиентах. После получения необходимых данных строятся графики о клиентах и операциях.

* добавление работников.

Необходимо заполнить соответствующую форму и нажать кнопку «Добавить», после чего данные занесутся в базу данных.

В процессе анализа аналогов ыло рассмотрено 2 системы: Инсайдер и TMetric. Обе системы разработаны для поддержки учета и анализа деятельности персонала на малых и крупных предприятиях и являются платными. Системы наделены широким функционалом, который упрощает многие бизнес-процессы. Среди преимуществ данных систем можно выделить: трекинг рабочих экранов, онлайн-мониторинг персонала, наличие мобильных версий приложений, интеграции с другими сервисами, анализ эффективности сотрудников, однако рассмотренные сервисы не предлагают расчет заработной платы, основываясь на собранных данных. Данную функцию я и решил добавить в свою программу, так как считаю, что упрощение ведения бухгалтерии – важный момент в программной поддержке предприятий.

## 1.4 Разработка информационной модели предметной области

MySQL – это система управления реляционными базами данных или СУБД. В реляционной базе данных данные хранятся в отдельных таблицах, благодаря чему достигается высокая скорость и гибкость работы. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных [2].

СУБД MySQL использует традиционную архитектуру клиент/сервер, поэтому, работая с СУБД MySQL, пользователь реально работает с двумя программами. - Программой сервера базы данных, расположенной на компьютере, где хранится база данных. Она «прослушивает» запросы клиентов, поступающие по сети, и осуществляет доступ к содержимому базы данных для предоставления информации, которую запрашивают клиенты. - Клиентской программой, которая является программой, осуществляющей подключение к серверу и передающей запросы на сервер.

В реляционной модели база данных представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. В строках таблиц часть полей содержит данные, относящиеся непосредственно к записи, а часть — ссылки на записи других таблиц. Таким образом, связи между записями являются неотъемлемым свойством реляционной модели.

В реляционной модели достигается информационная и структурная независимость. Записи не связаны между собой настолько, чтобы изменение одной из них затронуло остальные, а изменение структуры базы данных не обязательно приводит к перекомпиляции работающих с ней приложений.

В реляционных СУБД применяется язык SQL, позволяющий формулировать произвольные, нерегламентированные запросы. Это язык четвертого поколения, поэтому любой пользователь может быстро научиться составлять запросы. К тому же, существует множество приложений, позволяющих строить логические схемы запросов в графическом виде. Все это происходит за счет ужесточения требований к производительности компьютеров.

Процесс разработки (проектирования) базы данных включает два этапа: разработку логической организации базы данных и создание ее на носителе. Логическая организация базы данных – это предоставление пользователя о предметной области, информация о которой должна храниться в базе данных.

Под физической организацией базы данных понимается совокупность средств и методов размещения данных во внешней памяти и на их основе внутренняя модель данных. Внутренняя модель является средством отображения логической модели данных, показывает, каким образом записи размещаются в базе данных, как они упорядочиваются, как организуются связи, каким путем можно осуществить выборку и так далее.

Проектирование базы данных системы учета и анализа деятельности персонала на предприятии начиналось с создания всех нужных таблиц в базе, всех полей, входящих в каждую таблицу, взаимодействия таблиц между собой с помощью специальных отношений и создание в соответствии с этими параметрами первичных и вторичных ключей.

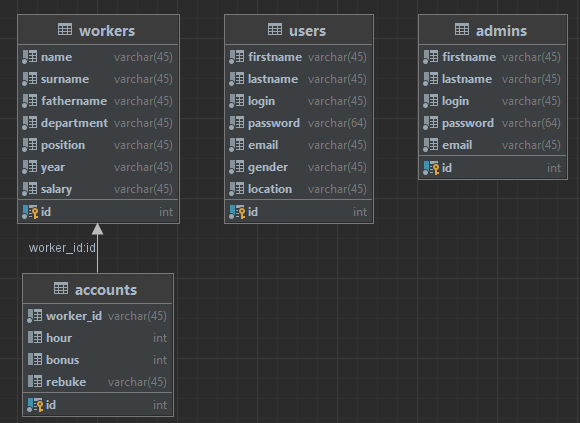


Рисунок 1.6 – Информационная модель базы данных

**1.5 UML-модели представления программного средства и их описание**

Первым шагом при описании функциональности системы является моделирование требований к ней.

Целями анализа и моделирования требований являются:

* достижение соглашения между разработчиками, заказчиками и пользователями о том, что должно делать ПС;
* достижение лучшего понимания разработчиками поведения ПС;
* ограничение системной функциональности;
* создание базиса для планирования разработки проекта;
* определение пользовательского интерфейса.

Для достижения этих целей используются диаграммы вариантов использования UML (Unified Modeling Language). На диаграммах вариантов использования (ВИ) изображаются актеры и варианты использования, между которыми существуют отношения. Актером будем называть внешнюю по отношению к ПС сущность, которая может взаимодействовать с системой. Актерами могут быть как люди, так и внешние системы или устройства. Следует всегда помнить, что актер – это не конкретный человек или устройство, а роль (должностная обязанность), в которой он выступает по отношению к программной системе.

Нахождение актеров – один из первых шагов в определении использования любой системы (как реальной, так и программной). Каждый источник внешних событий, с которыми должна взаимодействовать система, представляется как актер. Актер должен иметь имя, которое должно отражать его роль.

В данном случае актером является сотрудники предприятии.

В свою очередь, вариант использования (use case) служит для описания сервисов, которые предлагает система.

В разрабатываемой системе предполагается два актёра: администратор системы и клиент.

Система учета и анализа деятельности персонала на предприятии состоит из того, чтобы предоставить сотрудникам следующих действий:

* авторизация – пользователь вводит свой логин и пароль. Если введённый логин есть в базе данных, то дальше проверяем, совпадает ли пароль, введённый пользователем с паролем в базе данных;
* управления данными о персонале – сотрудники может добавлять, удалять, редактировать и просмотреть информацию о персонале предприятия;
* назначение премий и дисциплинарных взысканий персоналу;
* расчет заработной платы;
* составление отчетности по заработной плане персонала предприятия;
* составление статистики по выговорам в разных отделах предприятия;
* составление статистики по часам работы в разных отделах предприятия.

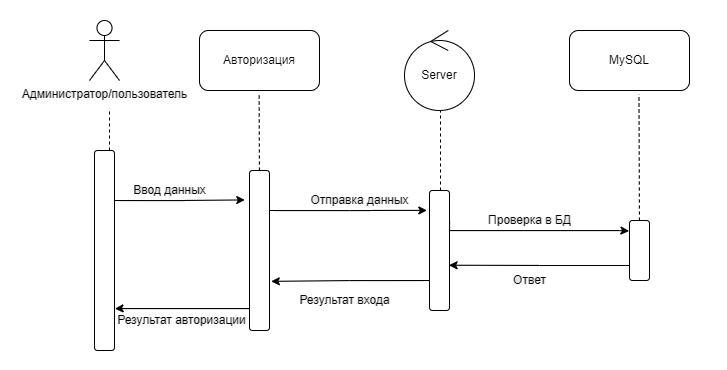


Рисунок 1.7 – Диаграмма последовательностей авторизации пользователя

# 2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Постановка задачи

Как известно основу любого предприятия составляет ведение учета и анализа. Автоматизации данных процессов программой поддержки экономит предприятиям время и деньги, а также уменьшает вероятность появления ошибки в отчетных документах, которая может доставить множество проблем.

Данный курсовой проект предполагает создание системы, которая соответствует следующим требованиям:

* хранение данных в персистентном хранилище;
* разделение ролей пользователей на обычного пользователя и администратора;
* предоставление удобного интерфейса для конечного пользователя;
* возможность добавления, редактирования, удаления записей;
* возможность регистрации новых пользователей.

Таким образом, информационная система для программной поддержки предприятия создаёт дополнительное пространство для ведения учета и автоматизации аналитических процессов бизнеса.

Приложение, созданное в качестве данного курсового проекта, написано на языке программирования Java. Java – объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Java поддерживает множество библиотек — строительных блоков любой корпоративной системы. Библиотеки помогают разработчикам создавать любые функции, которые могут понадобиться компании. Java широко распространен — это язык, который преподают в рамках введения в программирование в большинстве школ и университетов. Возможности интеграции Java впечатляют: большинство хостинг-провайдеров поддерживают Java. Более того, Java –  язык, дешевый в обслуживании: работать с Java можно с любого компьютера, вне зависимости от конкретной аппаратной инфраструктуры [3].

JavaFX представляет инструментарий для создания кроссплатформенных графических приложений на платформе Java. JavaFX позволяет создавать приложения с богатой насыщенной графикой благодаря использованию аппаратного ускорения графики и возможностей GPU. JavaFX предоставляет большие возможности по сравнению с рядом других подобных платформ, в частности, по сравнению со Swing. Это и большой набор элементов управления, и возможности по работе с мультимедиа, двухмерной и трехмерной графикой, декларативный способ описания интерфейса с помощью языка разметки FXML, возможность стилизации интерфейса с помощью CSS, интеграция со Swing и многое другое. На данный момент для создания графических приложений с помощью языка Java JavaFX представляет предпочтительный способ, который пришел на смену AWT и Swing.

Все данные приложения хранятся в базе данных. В качестве базы данных была выбрана MySQL. Это СУБД с открытым исходным кодом. База данных MySQL – это самая популярная в мире база данных с открытым кодом. Благодаря своей проверенной производительности, надежности и простоте использования база данных MySQL наиболее часто используется для веб-приложений [2].

## 2.2 Архитектурные решения

За архитектурные решения было взято использование архитектурного паттерна MVC и паттернов проектирования.

Паттерн MVC (Model/View/Controller) состоит из объектов трех видов. Модель – это объект приложения, а вид – экранное представление. Контроллер описывает, как интерфейс реагирует на управляющие воздействия пользователя [4].

Первым с клиентским запросом взаимодействует контроллер, контроллер разбивает данный запрос на элементы, далее он инициализирует объекты модели. После обработки данные отправляются на уровень представления.

Паттерн MVC – простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса.

Диаграмма взаимодействия компонентов для паттерна «MVC» изображена на рисунке 2.3



Рисунок 2.1 – Диаграмма для паттерна «MVC»

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

Алгоритм— представляет собой пошаговую процедуру, которая определяет набор инструкций, которые должны быть выполнены в определенном порядке, чтобы получить желаемый результат. Алгоритмы, как правило, создаются независимо от базовых языков, то есть алгоритм может быть реализован на нескольких языках программирования.

При запуске открывается главная окно, в котором пользователь должен пройти авторизацию. При успешной авторизации в зависимости от уровня доступа, пользователь получает разный набор возможностей для работы с программой. Если выполнен вход под администратором, то пользователь имеет доступ к данным администраторов, пользователей, работников, их данных и статистики. Если же была выполнена авторизация в качестве клиента, то функционал программы предоставляет возможность просматривать персонал, а также отчеты по заработной плате работников и анализ отделов по различным факторам.

Для проведения авторизации пользователь указывает его логин и пароль, после ввода этих данных пароль хешируется и производится соединение с БД и на сервер отправляется запрос выборки данных о пользователях. Далее в цикле проверяется совпадение введенных данных с поступающими данными из базы данных. В случае несовпадения вызывается окно ошибки и пользователю предлагается ввести данные повторно.

## 2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс - интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Задачи пользователя компьютерной программы заключаются в манипуляции с объектом и его свойствами – данными. В отличие от операторов, пользователи выполняют профессиональную задачу с иной психологической структурой действий, другими целями, объектом труда и операциями, ресурсами, иной социальной средой взаимодействия. Разнообразие ситуаций, в которых могут работать интерактивные программные системы, затрудняет для разработчика выбор целей, которым необходимо следовать для создания удачного интерфейса. Различные исследователи и организации-разработчики программного обеспечения приводят разные рекомендации, но основные из них следующие:

* простота. Простой интерфейс позволяет пользователю быстрее адаптироваться, уменьшает вероятность его ошибок, да и разработчику проще отладить такой интерфейс. Интерактивная система хороша, если интерфейс интуитивно понятен, то есть соответствует предметной области и стилю мышления пользователя. Интерфейс должен быть легким для освоения и не создавать перед пользователем преграду, которую он должен будет преодолеть, чтобы приступить к работе;
* дружественность. Интерфейс дружественный, если пользователь, работая с ним, не испытывает дискомфорта. У пользователя должно складываться впечатление, что он управляет процессом. Кроме того, графический интерфейс должен быть построен в соответствии с эргономическими требованиями: цвета экрана и элементов, их размер, композиция. Важен темп выполнения операций, который должен соответствовать естественному темпу человека, среднее время отклика и его дисперсия. Пользователь должен всегда знать, на какой стадии процесса он находится;
* естественность интерфейса. Естественный интерфейс — такой, который не вынуждает пользователя существенно изменять привычные для него способы решения задачи. Это, в частности, означает, что сообщения и результаты, выдаваемые приложением, не должны требовать дополнительных пояснений;
* функциональность. Хотя вычислительная система и бывает в некоторых организациях в роли большой игрушки, но чаще её пытаются использовать для дела, особенно в том случае, когда выполнение работы иными средствами менее эффективно. Функциональность системы обозначает наличие значительной эффективности в выполнении операций, что делает её использование рентабельным. Интерфейс должен отражать ее функциональность и давать возможность успешной работы пользователям различной квалификации.

В данном приложении для разработки пользовательского интерфейса использовались элементы библиотеки JavaFX и Scene Builder, как инструмент для построения визуала [5].

## 2.5 Обоснование выбора компонентов и технологий для реализации программного средства

При проектирования данного приложения были использованы различные библиотеки, не включенные в требования. В этой главе будут рассмотрены некоторые из них.

Для хеширования паролей была выбрана библиотека Commons Codec, позволяющая быстро и эффективно хешировать строки через SHA-256.

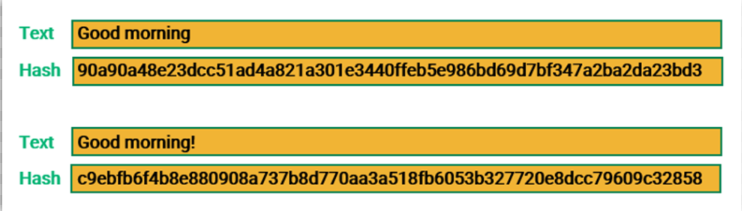


Рисунок 2.2 – Пример хеширование по алгоритму SHA-256

Для подключения к базе данных использовался драйвер JDBC. JDBC — это Java Database Connectivity API. Это старый стандарт, разработанный еще в 1997 году. Абсолютно все библиотеки, взаимодействующие с базами данных, используют JDBC. По сути, JDBC — это Java-реализация стандартного подхода к работе с базами данных, что и стало решающим фактором при выборе технологии для работы с базой данных. В коде используется SQL (Structured Query Language).

# 3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Для корректной работы приложения было предусмотрено ряд технических средств для проверки и тестирования программы. Пользователь своевременно получает сообщение об ошибках или неправомочных действиях. Рассмотрим критические случаи работы нашего приложения.

На сервере также ведётся лог действий пользователя (рисунок 3.1).

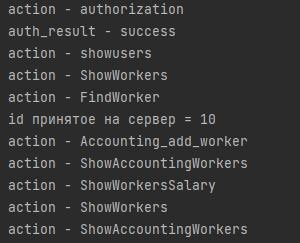


Рисунок 3.1 – Логи сервера

В окне регистрации пользователю необходимо заполнить данные о сервере, логин и пароль. Если же пользователь ввел невалидные значения, клиент получает соответствующее сообщение в логах (рисунок 3.2, рисунок 3.3).



Рисунок 3.2 – Сообщение о неверных данных сервера

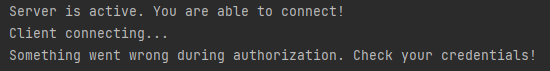


Рисунок 3.3 – Сообщение о неверных данных пользователя

# 4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ И СКВОЗНОЙ ТЕСТОВЫЙ ПРИМЕР, НАЧИНАЯ ОТ АВТОРИЗАЦИИ, ДЕМОНСТРИРУЯ РЕАЛИЗАЦИЮ ВСЕХ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

При запуске приложения появляется окно авторизации, в котором мы можем ввести данные аккаунта и параметры подключения к серверу такие как ip с и порт для входа в приложение.

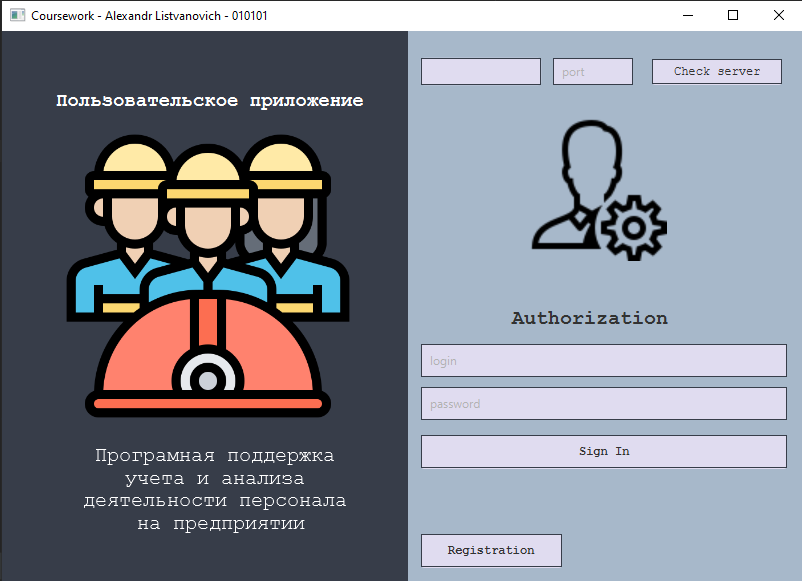


Рисунок 4.1 – Форма авторизации пользователя

Если же у пользователя нету аккаунта он может нажать на кнопку регистрации и тогда откроется окно регистрации где пользователь может зарегестрировать новый аккаунт.

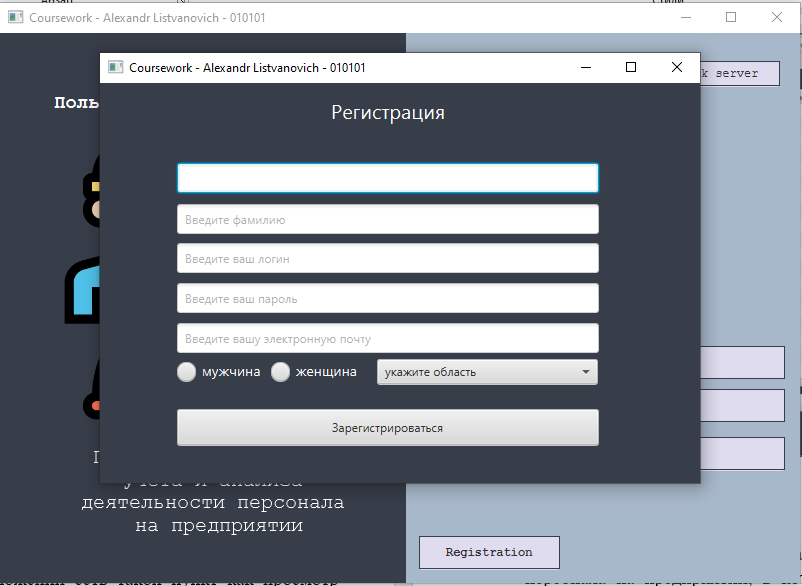


Рисунок 4.2 – Форма регистрации пользователя

После прохождения авторизации пользователю будет открыто пользовательское приложение.

В пользовательском приложении есть такой пункт как просмотр персонала. В этом пункте есть таблица с персоналом и кнопка обновить которая обновляет таблицу если администраторы внесли изменение в список сотрудников. Рис 4.3

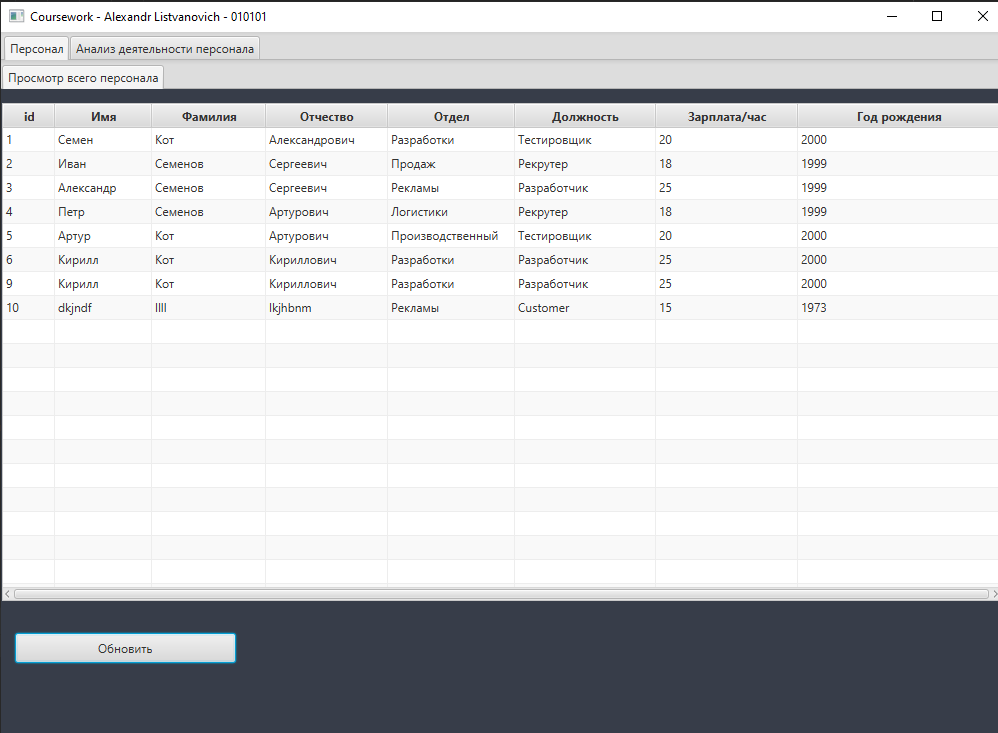


Рисунок 4.3 – Просмотр персонала

В приложении пользователя есть раздел аналитики деятельности персонала на предприятии, в котором пользователь может получить всю основную информацию об работе предприятия наглядно на круговых диаграммах и таблицах, так же пользователь может рассчитать зарплаты сотрудников и общую итоговую сумму денег которое предприятие должно своим сотрудникам.

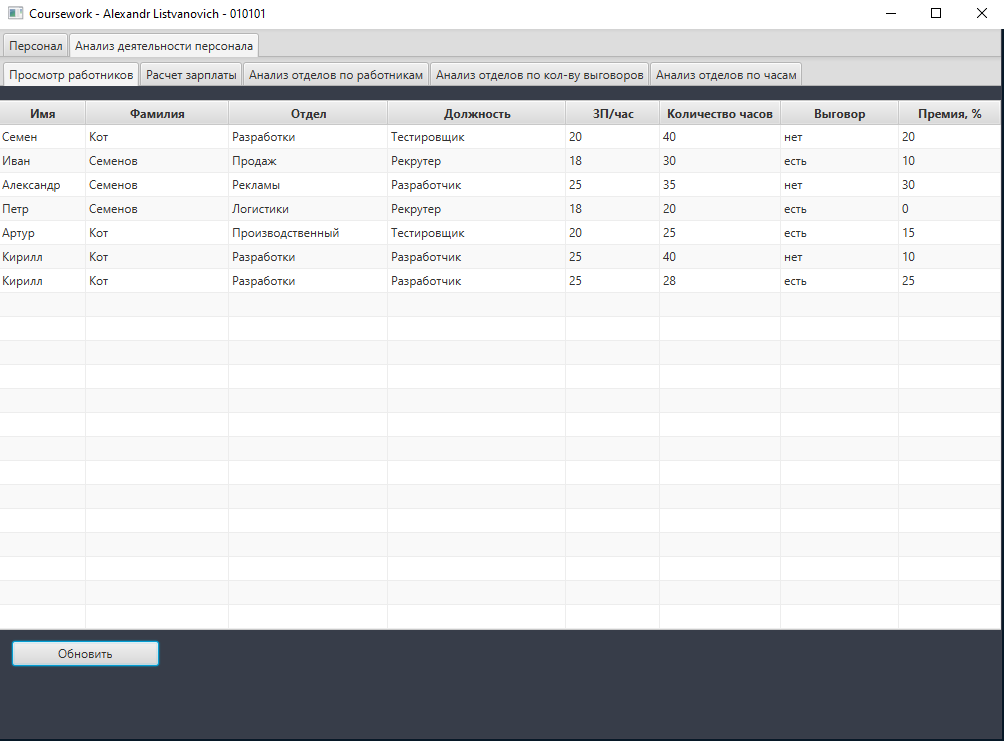


Рисунок 4.4 – Просмотр работников

На рисунке 6,4 пользователь может посмотреть всех сотрудников которые внесены в учет, узнать количества отработанных часов, наличие премии, наличие дисциплинарного взыскания. Так же пользователь может обновить статистику если администратор приложения внесет изменения в таблицу.

Далее идет вкладка «Расчет заработных плат сотрудникам» в которой пользователь может получить подробную информацию о заработных платах сотрудников и итоговую сумму зарплат.

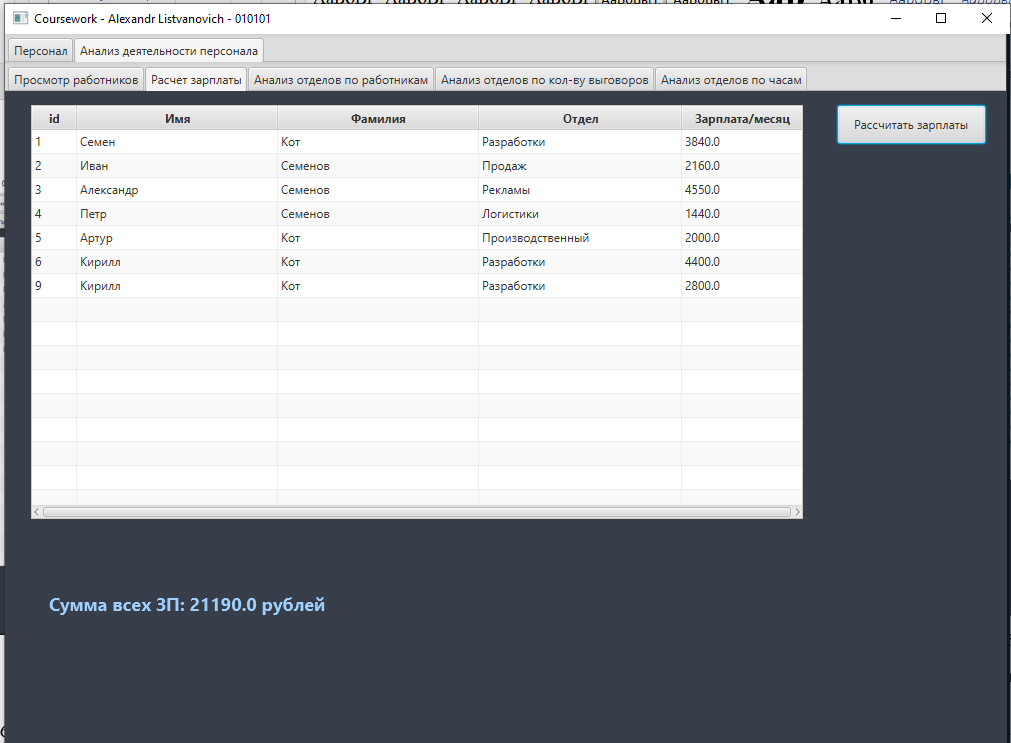


Рисунок 4.5 – Расчет заработной платы

В данной таблице предоставлена информация о зарплатах сотрудникам

Далее пользователь может посмотреть более наглядную статистику по работе предприятия на круговых диаграммах.

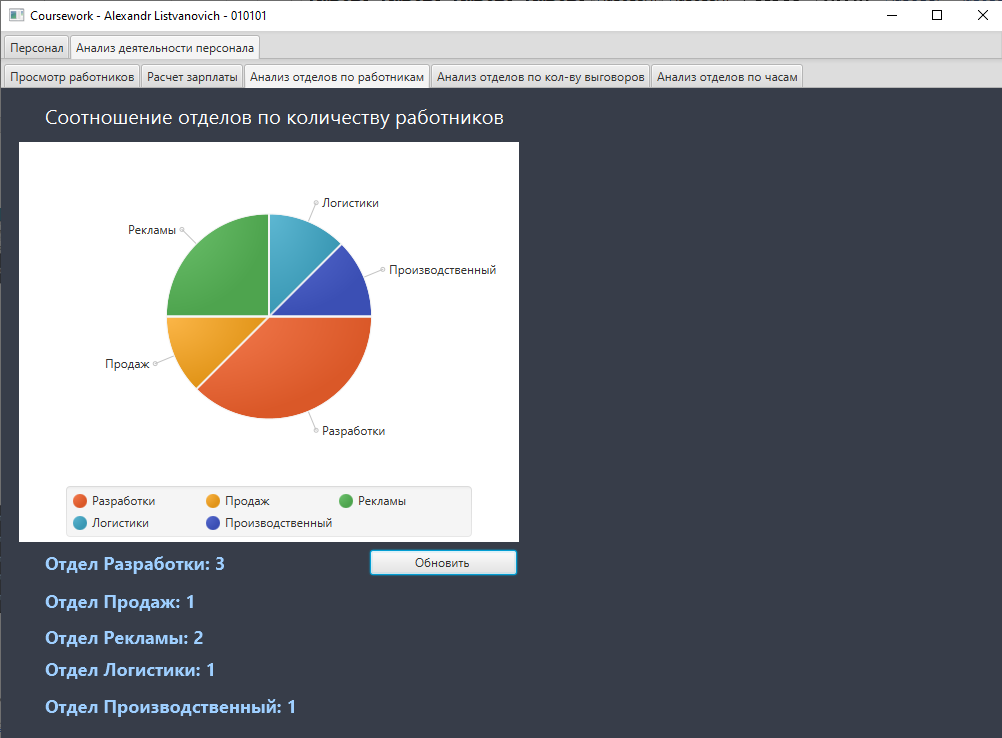


Рисунок 4.6 – Анализ отделов по количеству работников

На данной круговой диаграмме (Рисунок 4.6) представлена статистика по отделам которая показывает сколько работников работает в каждом отделе, другими словами предоставляет информацию о размере отделов. Так же в данной вкладке есть кнопка «Обновить» которая обновляет статистику если администратор приложения внес изменения в таблицу.

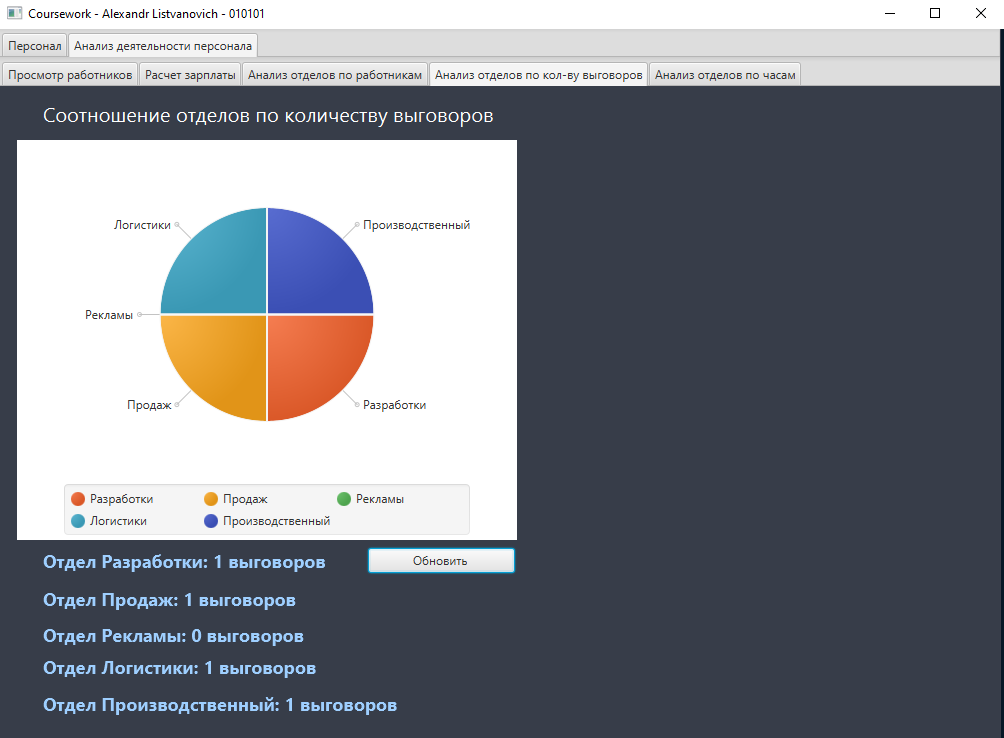


Рисунок 4.7 – Анализ отделов по количеству выговоров

На рисунке 4.6 переде нами предстает круговая диаграмма которая показывает информацию о соотношении отделов количеству дисциплинарных взысканий а под данной круговой диаграммой располагается информация в текстовом виде которая отражает количественное соотношение дисциплинарных взысканий на каждый отдел. Также под круговой диаграммой располагается кнопка «обновить» при нажатии на которую будет обновлена статистика если администратор ввел изменения в учет деятельности сотрудников предприятия.

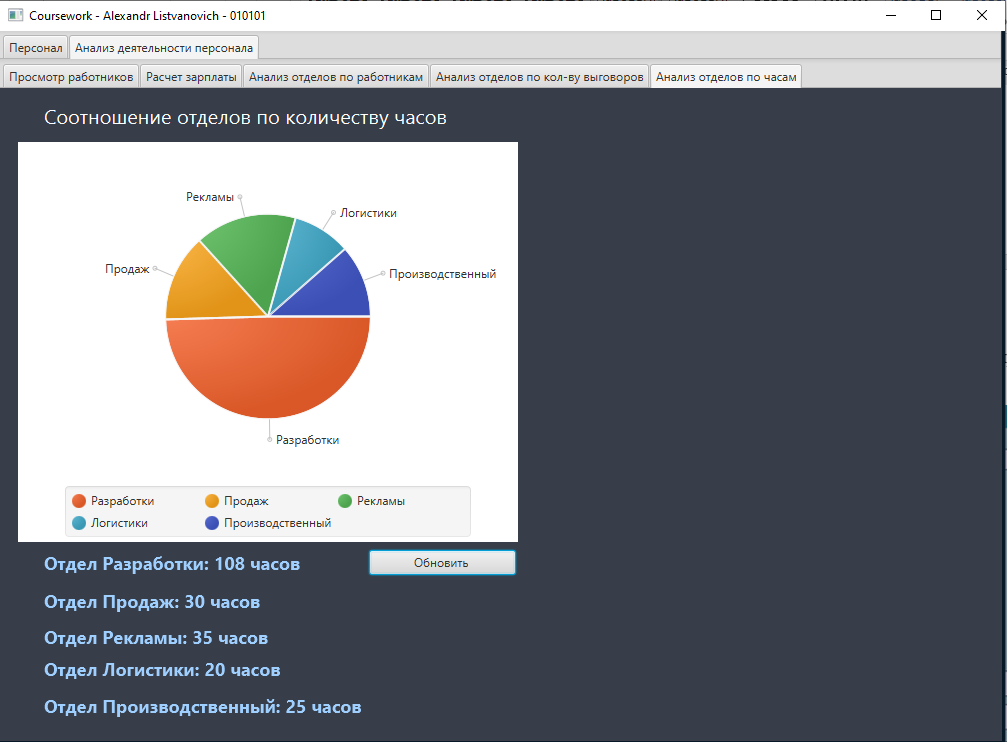


Рисунок 4.8 – Анализ отделов по количеству рабочих часов

Во вкладке «анализ отделов по часам» мы можем посмотреть соотношение отделов по количеству отработанных сотрудниками часов. Мы можем увидеть круговую диаграмму, которая визуально показывает нам соотношение отделов, а под данной круговой диаграммой мы можем увидеть суммарное количество отработанных часов всеми сотрудниками определенного отдела, которых администратор приложения добавил в учет. Также под диаграммой есть кнопка «обновить» которая обновляет нам статистику, если администратор приложения внес изменения в учет деятельности сотрудников.

При запуске приложения администратора нас встречает окно авторизации в котором администратор должен указать логин и пароль, а также данные для подключения к серверу такие как ip адрес и порт. В данном окне отсутствует кнопка регистрации т.к. регистрировать администратора могут только другие администраторы. (Рисунок 4.9)

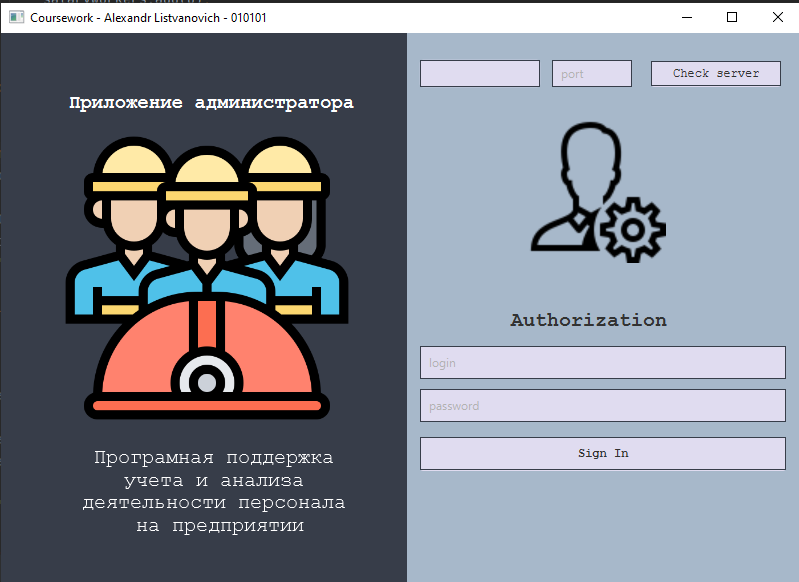


Рисунок 4.9 – Форма авторизации администратора

После успешной авторизации в приложении администратору открывается администраторское приложение. В котором администратора встречает вкладка «Работа с администраторами».

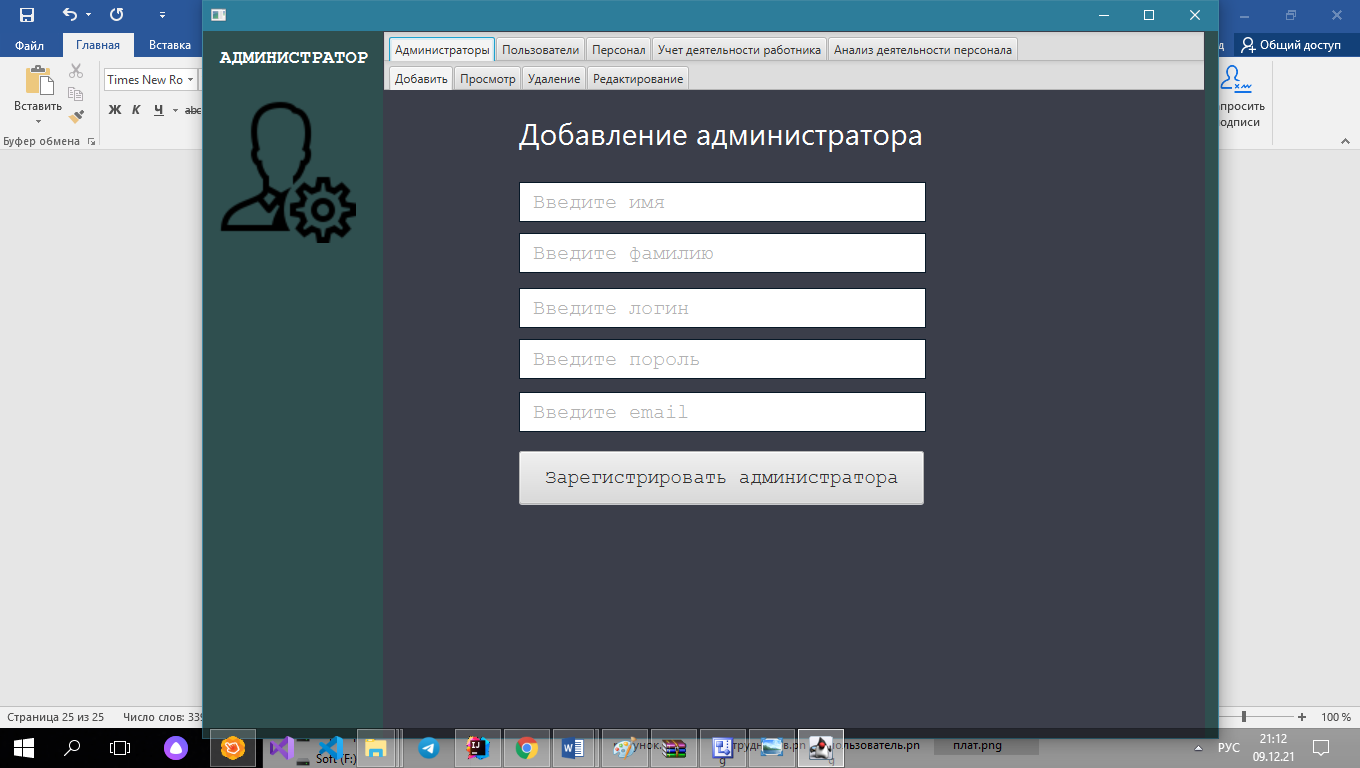


Рисунок 4.10 – Форма добавления администратора

Во вкладке «Администраторы» имеется интерфейс добавления нового администратора. Для добавления нового администратора нужно ввести имя, фамилию, логин, пароль, и электронную почту будущего администратора. Далее нужно нажать на кнопу «Зарегистрировать администратора» и после этого новый администратор будет добавлен в базу данных администраторов.

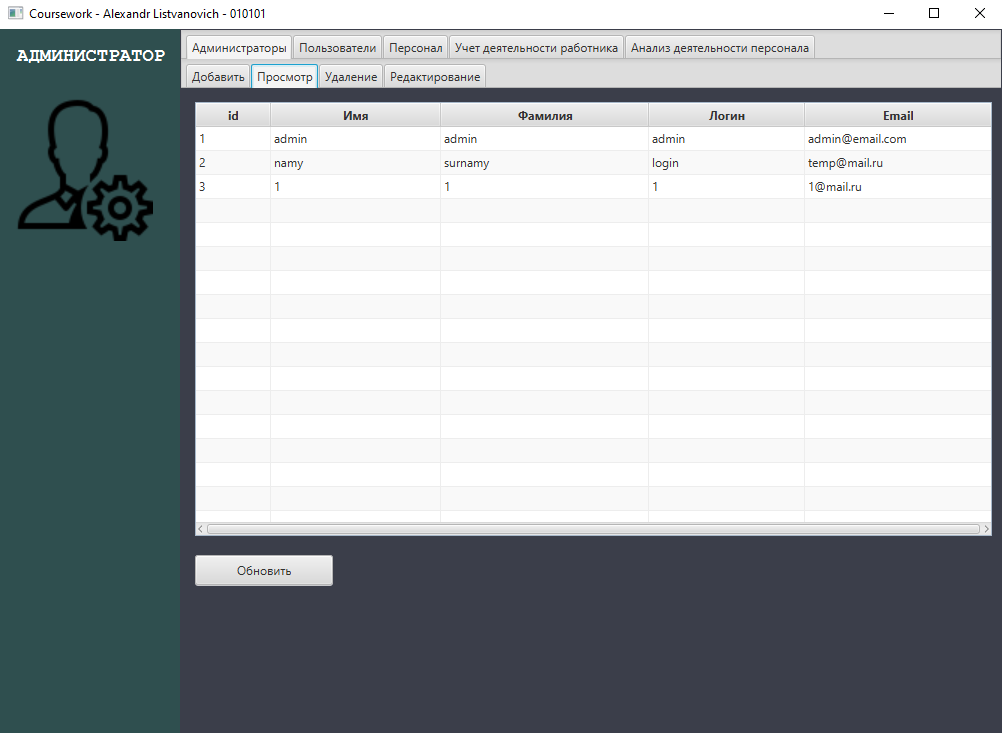


Рисунок 4.11 – Просмотр администраторов

Окно «просмотр» предназначено для просмотра уже существующих администраторов здесь можно узнать имя, фамилию, логин, и электронную почту других администраторов. При нажатии на кнопку обновить таблица администраторов будет обновлена если в нее внесены соответствующие изменения.

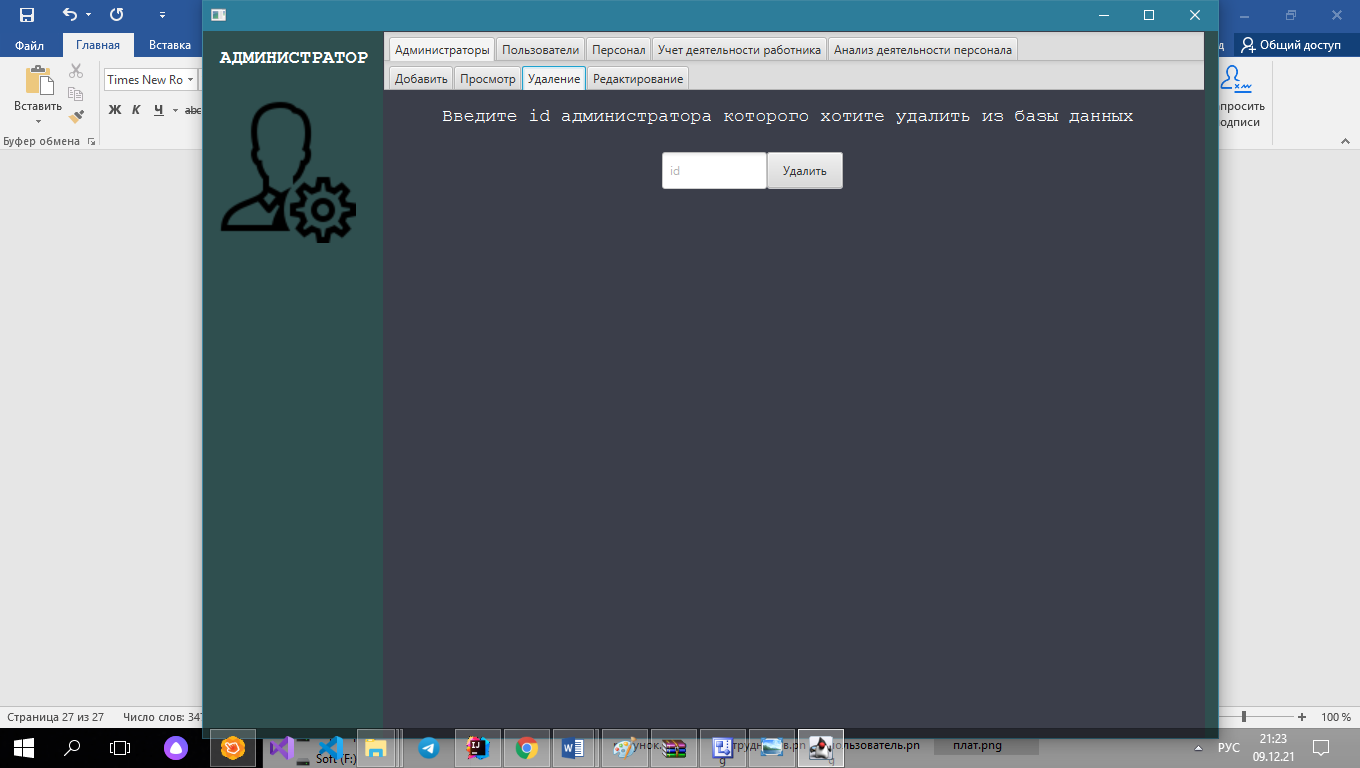


Рисунок 4.12 – Удаление администратора

Во вкладке «Удаление администратора» можно удалить других администраторов и самого себя, но при этом зайти в приложение уже будет нельзя. Для удаления администратора необходимо ввести идентификационный номер администратора и нажать на кнопку «Удалить» и если администратор с таким идентификационным номером существует, то он будет удален, чтобы узнать результат удаления администратора нужно зайти во вкладку просмотр администраторов (Рисунок 4.11) и нажать на кнопку «Обновить».

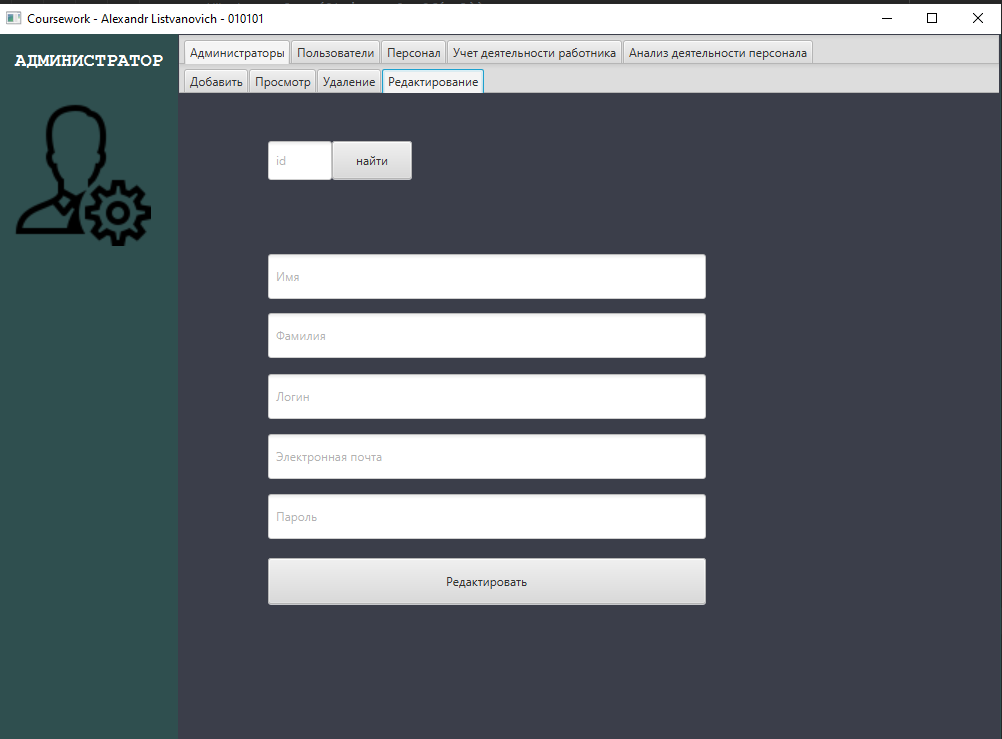


Рисунок 4.13 – Форма редактирования администратора

Во вкладке редактирование администратора нам доступна функция редактирования администратора, в которой пользователю необходимо ввести идентификационный номер администратора, которого он хочет отредактировать и нажать кнопку найти далее будет предоставлена текущая информация, об администраторе которую мы можем отредактировать. Также администратор может редактировать и себя. После нажатия на кнопку администратора администратор будет отредактирован, чтобы узнать результат редактирования необходимо перейти во вкладку просмотр администраторов (Рисунок 4.11) и нажать на кнопку обновить.

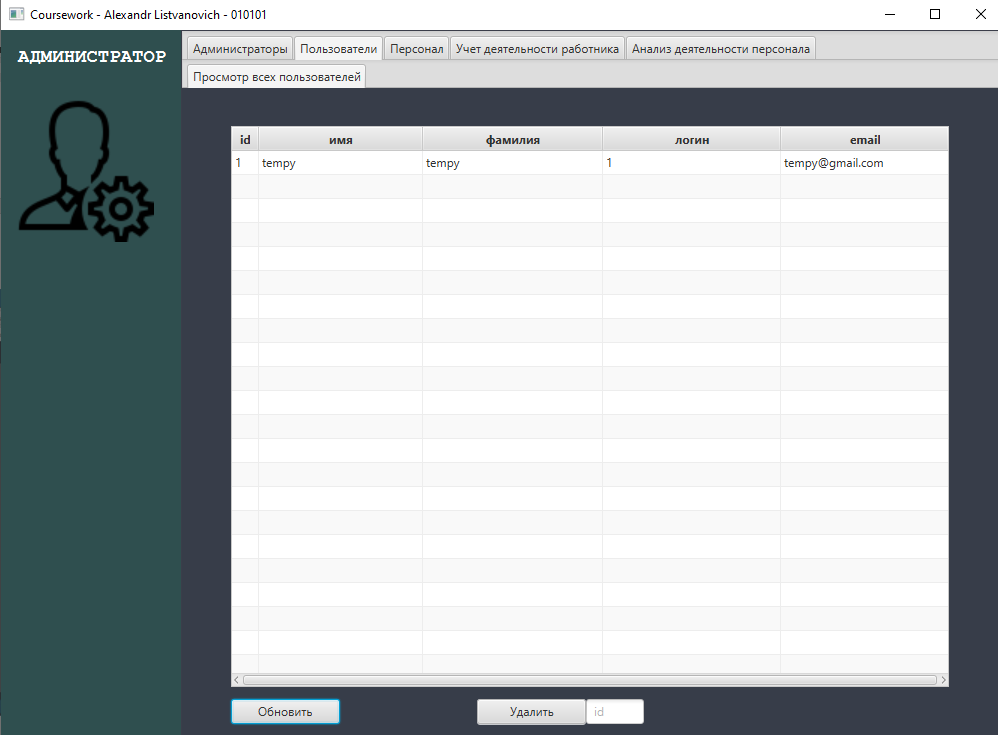


Рисунок 4.14 – Просмотр пользователей

Раздел «Пользователи» включает в себя только одну вкладку «Просмотр всех пользователей» которая объединила в себе функции просмотра и удаления аккаунтов пользователей. Перед администратором предстает таблица где можно увидеть информацию об аккаунтах пользователей, под данной таблицей находится две кнопки «обновить» и кнопка «удалить». При нажатии на кнопку обновить информация об аккаунтах пользователей будет обновлена если администратор удалил пользователя либо же новый пользователь зарегистрировался. Для удаления аккаунта пользователя администратору необходимо ввести идентификационный номер аккаунта пользователя в поле ввода идентификационного номера аккаунта пользователя, далее администратору необходимо нажать на кнопку «удалить» и тогда аккаунт выбранного администратором пользователя будет удален.

Одним из самых важных разделов данного приложения является раздел, «Работники» который предоставляет администратору возможность добавлять нового работника в базу данных, просматривать всех уже добавленных работников, удалить работника из базы данных и редактировать информацию о работнике.

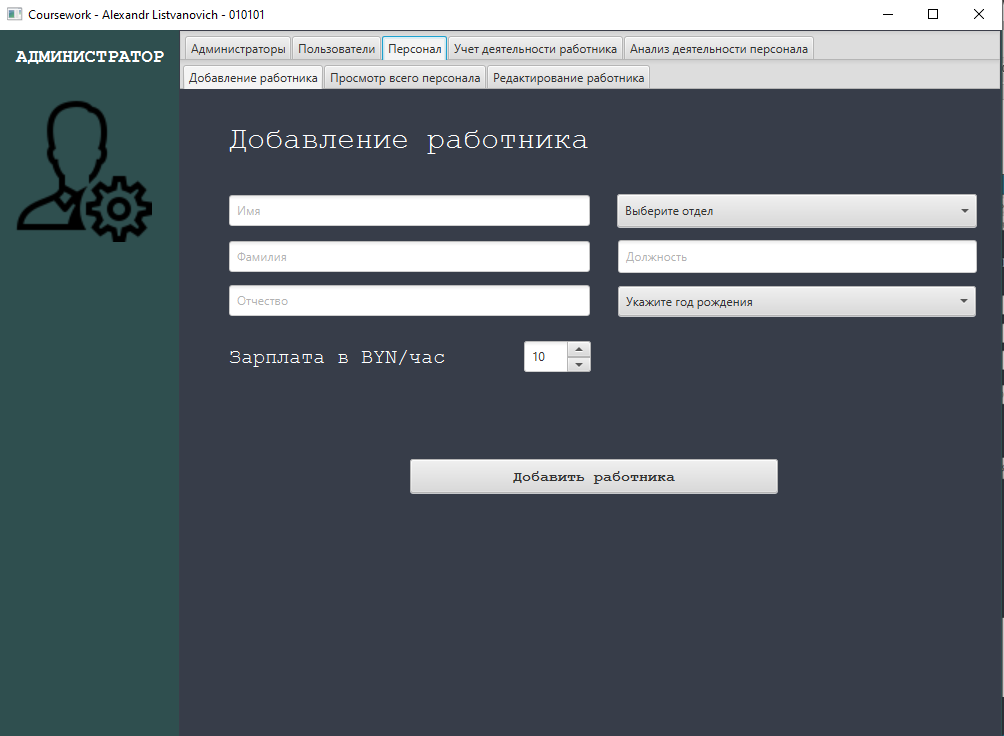


Рисунок 4.15 – Форма добавления работника

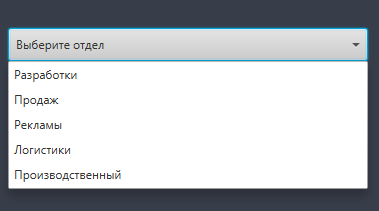


Рисунок 4.16 – Выбор отдела

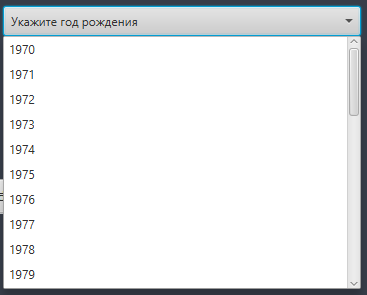


Рисунок 4.17 – Выбор года рождения

Для добавления работника администратор должен указать имя, фамилию, отчество, выбрать отдел из пяти предложенных (Разработки, Продаж, Рекламы, Логистики, Производственный) Рисунок4.14. Далее администратору необходимо указать должность и год рождения работника. Далее нужно указать заработную плату работника в час которая в дальнейшем будет использована для расчета заработной платы. После того как администратор введет все параметры работнику необходимо нажать на кнопку «Добавить работника» после чего работник будет добавлен в базу данных.

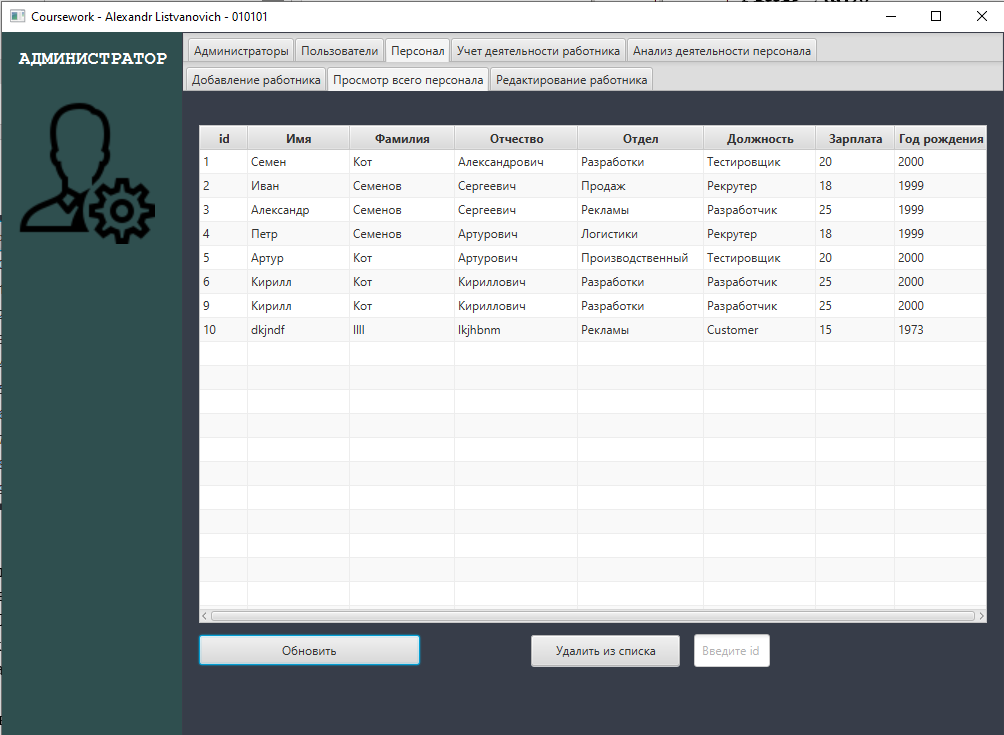


Рисунок 4.18 – Просмотр всего персонала

Во вкладке просмотр всего персонала можно получить подробную информацию о всех работниках, добавленных в базу данных, для этого необходимо нажать на кнопку обновить которая обновляет таблицу работников если данный либо же другой администратор внес в нее изменения. Под таблицей находится кнопка «Удалить из списка». Для удаления работника из таблицы необходимо ввести идентификационный номер работника в поле ввода идентификационного номера работника и нажать кнопку «удалить работника из списка» после чего работник будет удален, для проверки успешности удаления необходимо обновить таблицу работников нажав на кнопку «обновить».

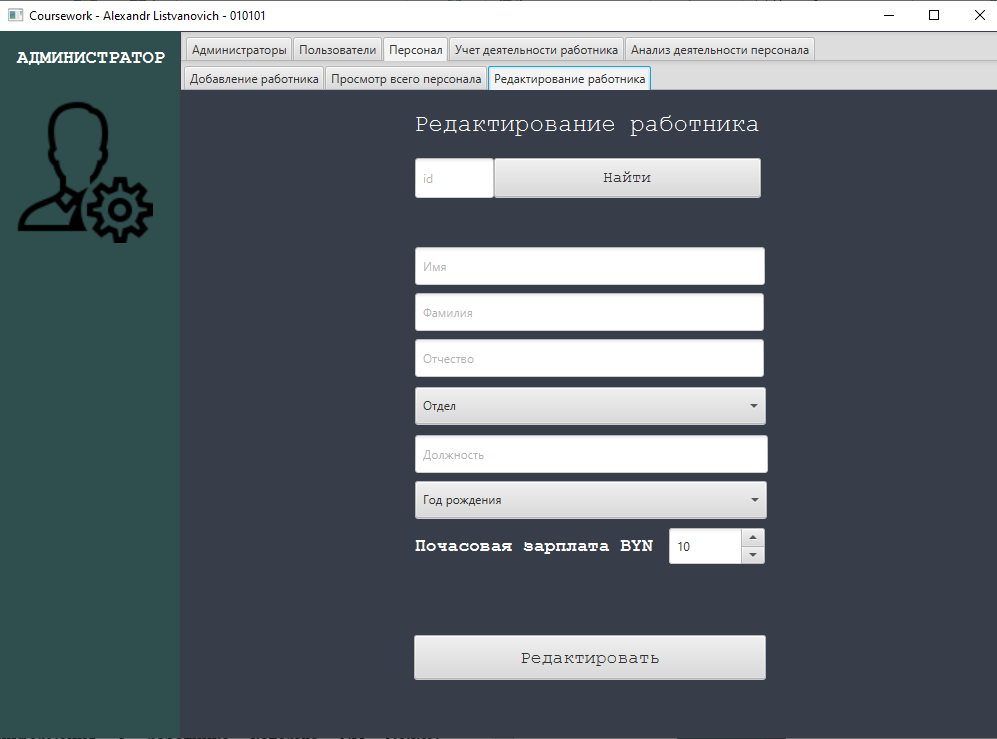


Рисунок 4.19 – Форма редактирования работника

Во вкладке «Редактирование работника» администратору нужно выбрать работника которого он хочет редактировать, для этого нужно ввести идентификационный номер работника в поле ввода идентификационного поля работника и нажать на кнопку «Найти» после чего в полях редактирования появится текущая информация о работнике которую мы можем отредактировать, после редактирования полей информации о работниках необходимо нажать на кнопку «Редактирование» после чего информация о работнике будет отредактирована и перезаписана в базу данных.

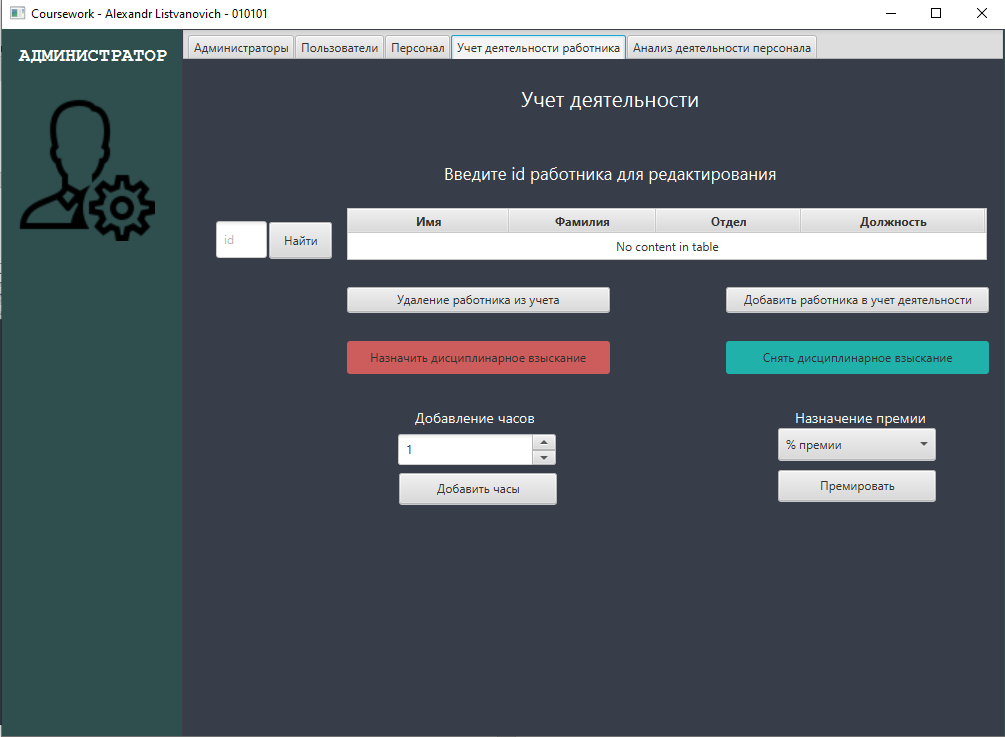


Рисунок 4.20 – Меню учета деятельности работника

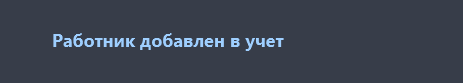


Рисунок 4.21 – Сообщение об изменении в учете деятельности работника

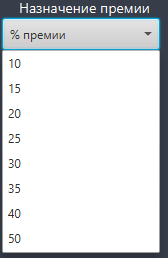


Рисунок 4.22 – Выбор объёма премии работнику

Вкладка «Учет деятельности работника» является ключевой в этом приложении. В данной вкладке ведется учет деятельности работника, на основании которого будет вестись расчет заработной платы сотрудникам и анализ деятельности сотрудников на предприятии. Эта вкладка включает в себя такие функции как добавление работника в учет, удаление работника из учета, добавление часов работнику, назначение премии, назначение дисциплинарного взыскания работнику и снятие уже имеющегося дисциплинарного взыскания. Для работы с учетом деятельности работника в первую очередь администратору необходимо ввести идентификационный номер работника в поле ввода идентификационного номера работника после чего необходимо нажать на кнопку «выбрать» после чего в таблице будет показана текущая информация о работнике имя, фамилия, отдел и должность. После выбора работника администратор может добавить работнику отработанные часы. Для этого необходимо выбрать количество часов, которое он хочет назначить работнику от 1 до 12, и нажать на кнопку добавить часы после чего в поле вывода текущего состояния внизу экрана будет выведено текущее состояние. Кнопка «премировать» предназначена для начисления премии сотруднику, для этого необходимо выбрать размер премии (в процентах) Рисунок 4.22. и нажать на кнопку премировать. Также в данной вкладке присутствует две кнопки красная и зеленая. При нажатии на красную кнопку работнику будет выдано дисциплинарное взыскание, а при нажатии на зеленую с работника будет снято текущее дисциплинарное взыскание если такое имеется.

Вкладка «анализ деятельности персонала на предприятии» аналогична той, которая имеется в пользовательском приложении.

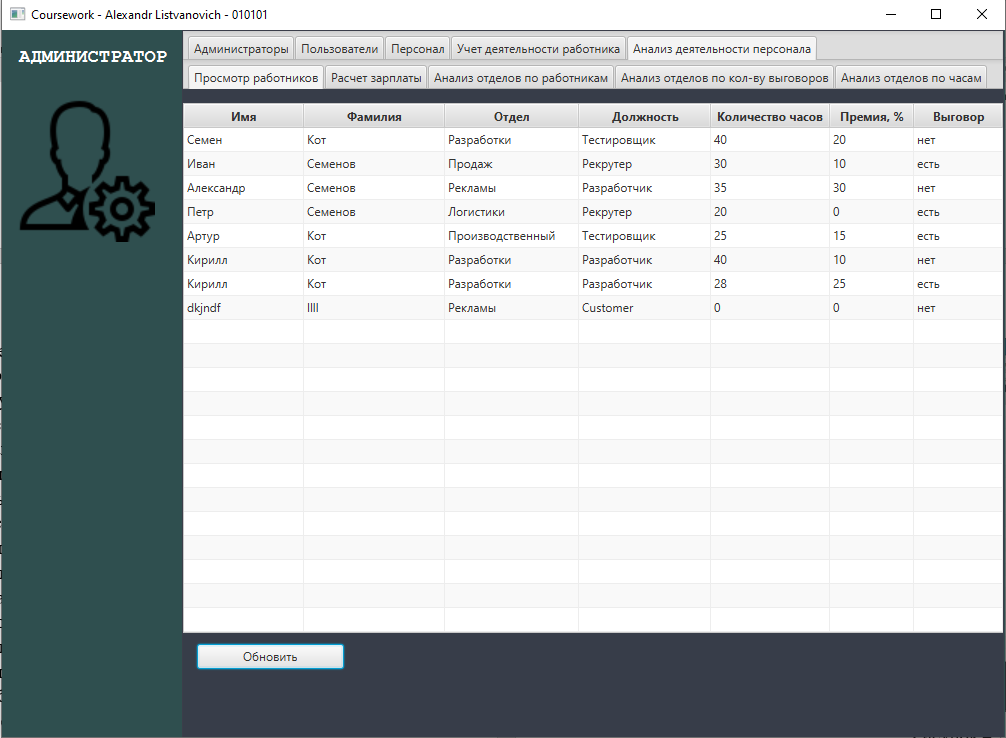


Рисунок 4.23 – Просмотр всех работников

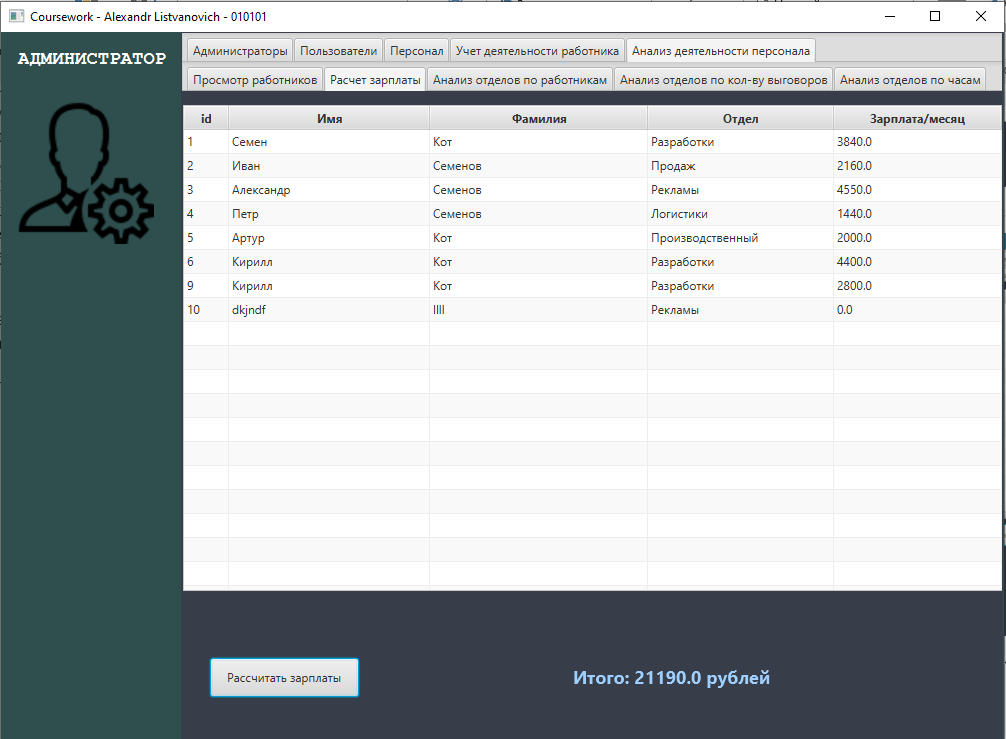


Рисунок 4.24 – Меню расчета заработной платы

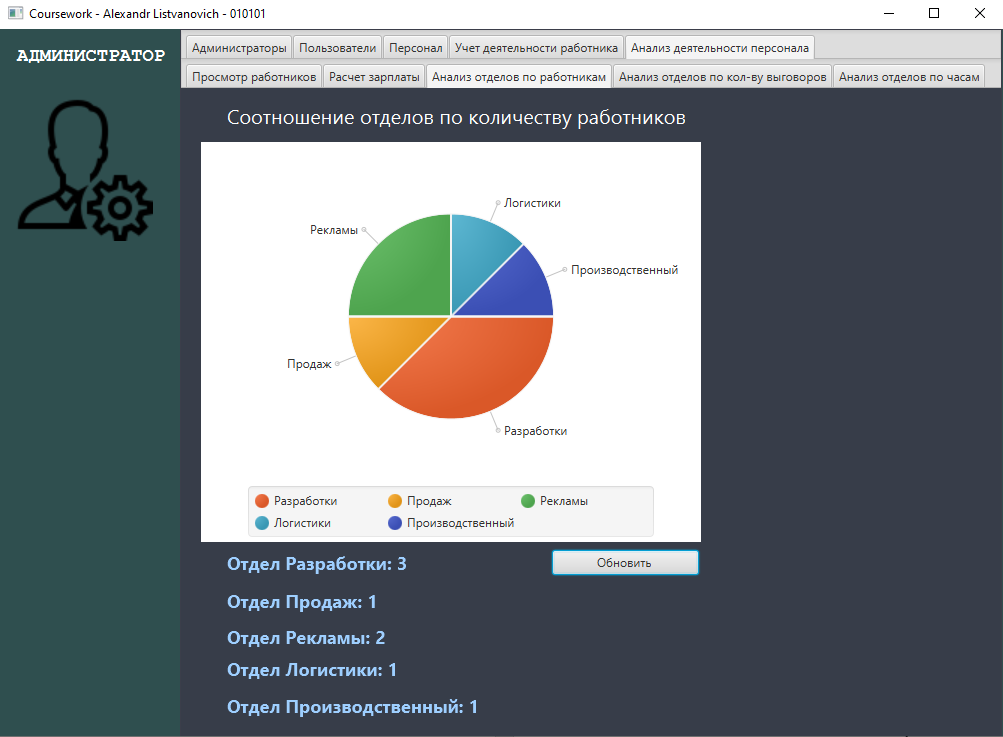


Рисунок 4.25 – Анализ отделов по количеству работников

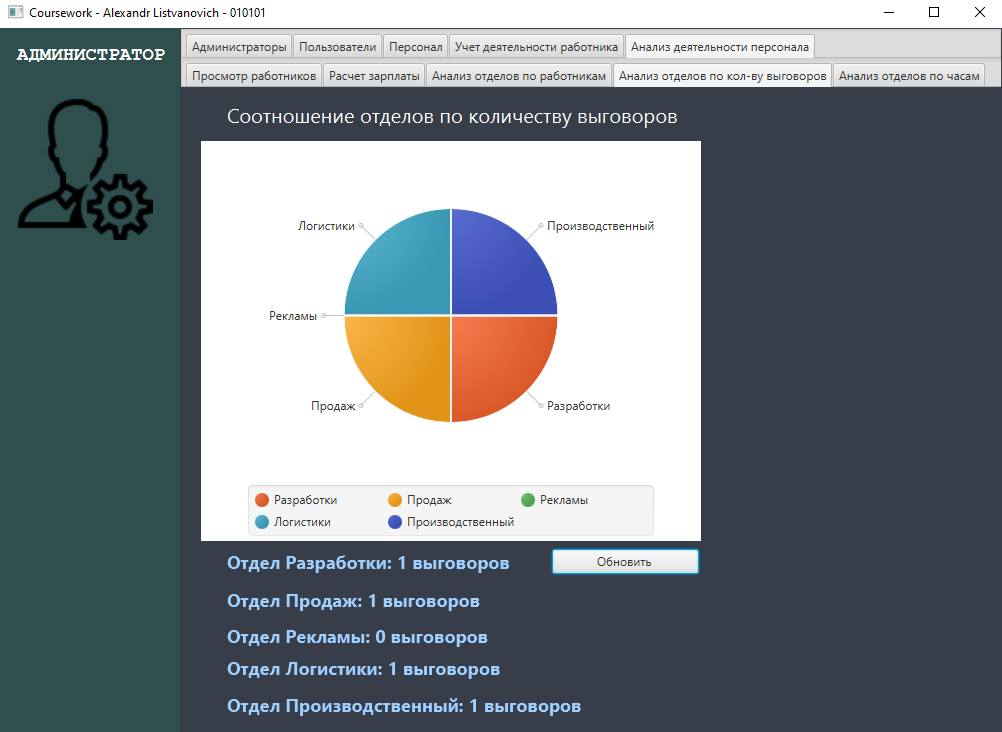


Рисунок 4.26 – Анализ отделов по количеству выговоров

# ЗАКЛЮЧЕНИЕ

Прогресс, достигнутый за последние несколько лет во всех аспектах вычислительной техники, включая теорию, технологию и приложения, привели к значительному расширению области применения компьютеров и росту числа их пользователей. Существенной частью современного общества являются разнообразные системы доступа и хранения информации, которые являются неотъемлемой составляющей современного научно-технического прогресса. Существует много веских причин перевода существующей информации на компьютерную основу, так как более быстрая обработка данных и централизация их хранения с использованием клиент/серверных технологий позволяют сберечь значительные средства, а главное и время для получения необходимой информации, а также упрощает доступ и ведение.

В результате проделанной работы был проведен анализ предметной области, а именно системы управление тур фирмы.

При этом задачи, поставленные перед началом выполнения, были достигнуты следующим образом:

* в ходе программы реализован множественный функционал, в разной степени доступный пользователям в зависимости от их прав;
* программный продукт позволяет сотрудникам записать данные
* данный продукт может быть использован, как сотрудниками клиники, так и простыми пользователями;
* предусмотрена обработка ошибок, которые могут возникнуть в процессе работы;
* применение базы данных в качестве хранилища информации позволяет оптимально и эффективно хранить информацию, ее структурировать;
* безопасность обеспечивается за счет разграничения прав доступа, системы авторизации;
* для удобства пользователя, программа располагает достаточно понятным, наглядным и удобным интерфейсом взаимодействия пользователя с программой;
* интерфейс программы поддерживает русский язык.

Данное приложение позволит повысить производительность и качество работы туристической фирмы, поможет работникам тур фирмы в решении практических задач, позволит снизить трудозатраты, улучшит качество и повысит скорость обслуживания клиентов.

В процессе разработки, приложение было протестировано, в результате не было выявлено фатальных ошибок, а все выявленные незначительные ошибки были устранены.

Все поставленные задачи решены. В будущем приложение будет совершенствоваться посредством улучшения и расширения информационной модели системы, оптимизации имеющихся функций и добавления новых. Также будет улучшен интерфейс приложения.

Помимо бизнес-логики также были определены требования к работе, внешнему виду и содержанию системы, а также были определены основные сущности системы для организации моделей.

Подводя итог всему выше сказанному, можно утверждать, что разработка таких программ позволяет значительно оптимизировать и упростить деятельность подобных учреждений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://studme.org/87184/ekonomika/metodologiya\_idef0/.

[2] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://www.mysql.com/products/workbench/.

[3] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://en.wikipedia.org/wiki/Java>/.

[4] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://studfile.net/preview/8839520/page:6/.

[5] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://openjfx.io/.

# Приложение А

# (обязательное)

# UML Диаграммы классов

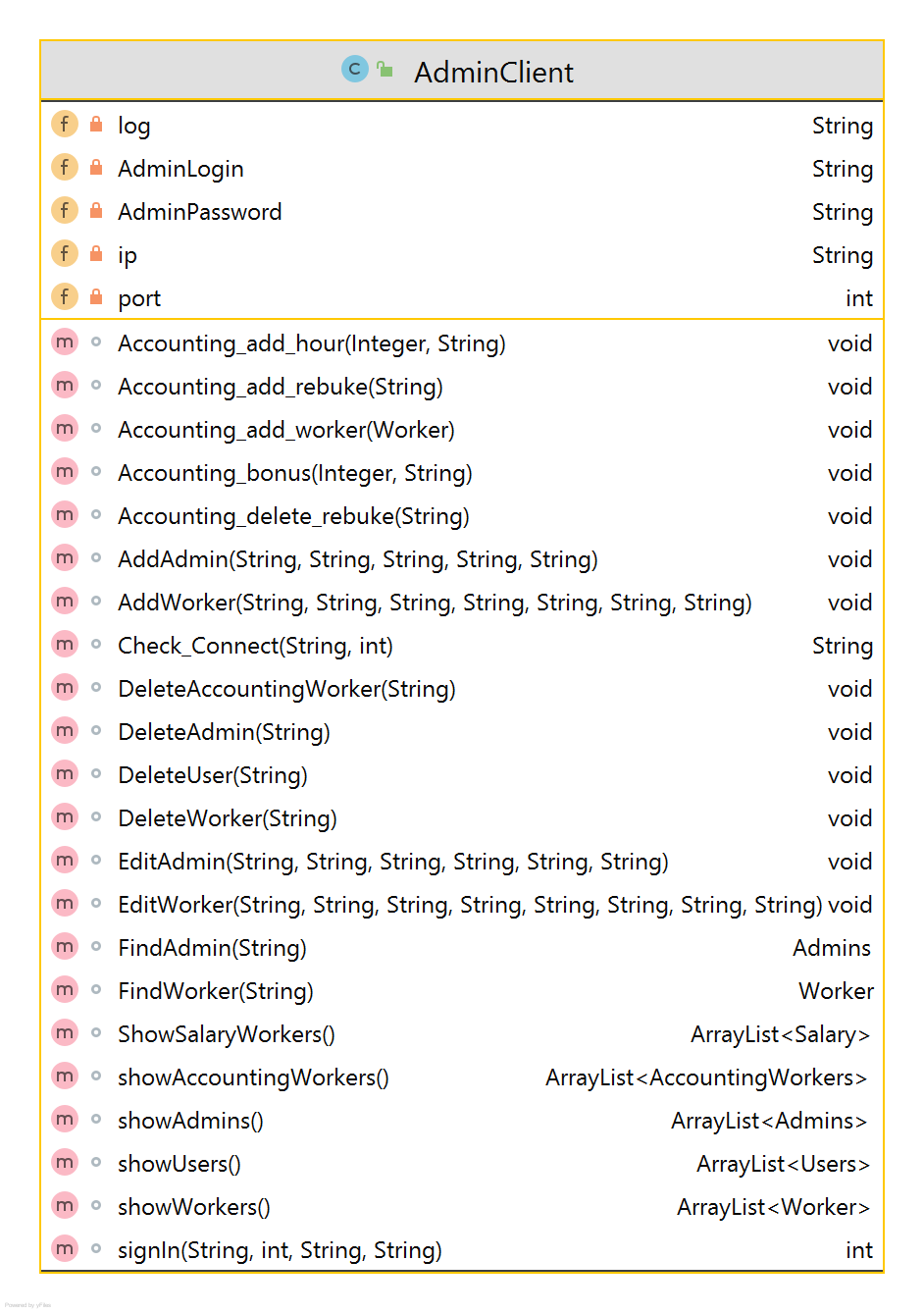


Рисунок А.1 – Класс AdminClient

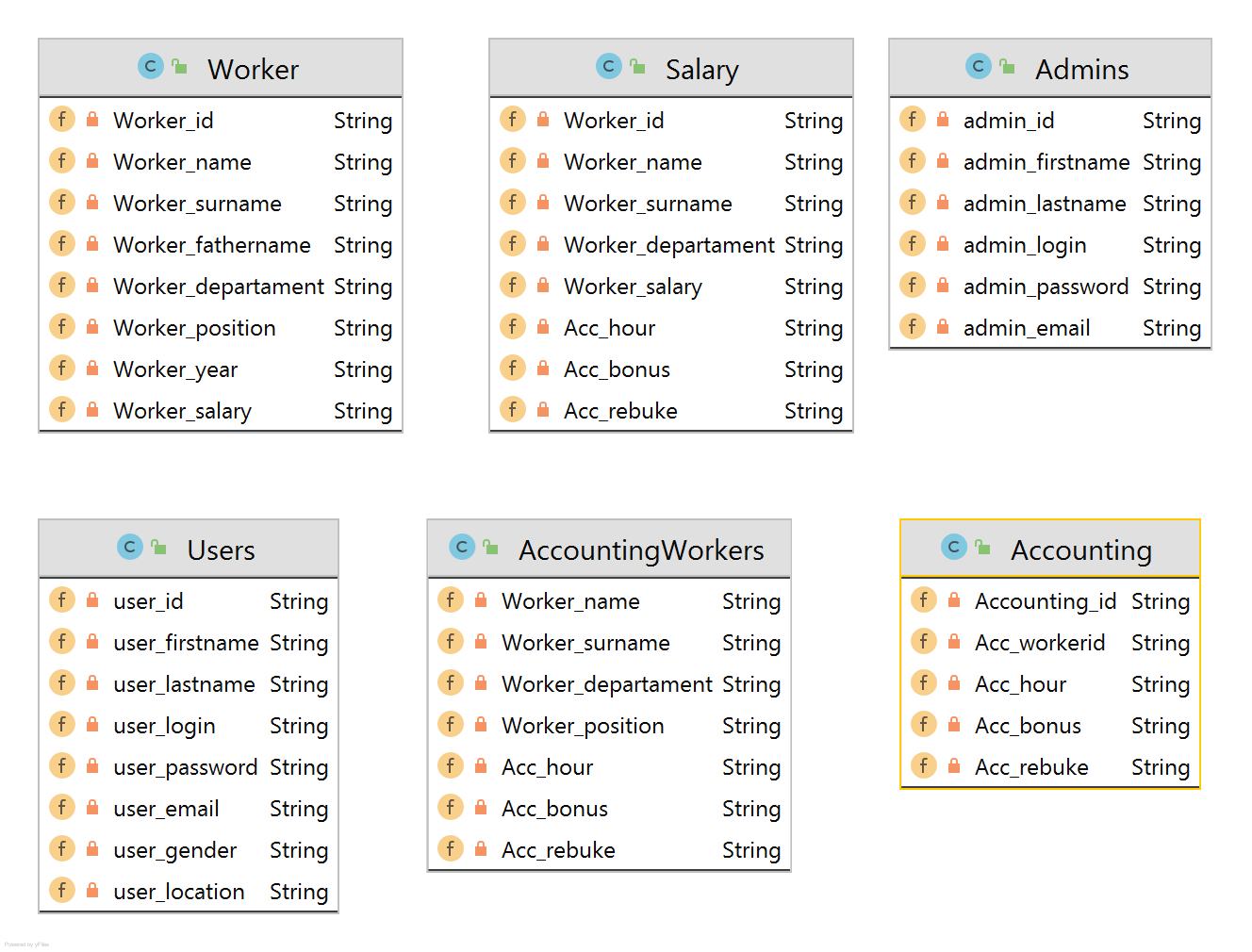


Рисунок A.2 – Основные классы данных

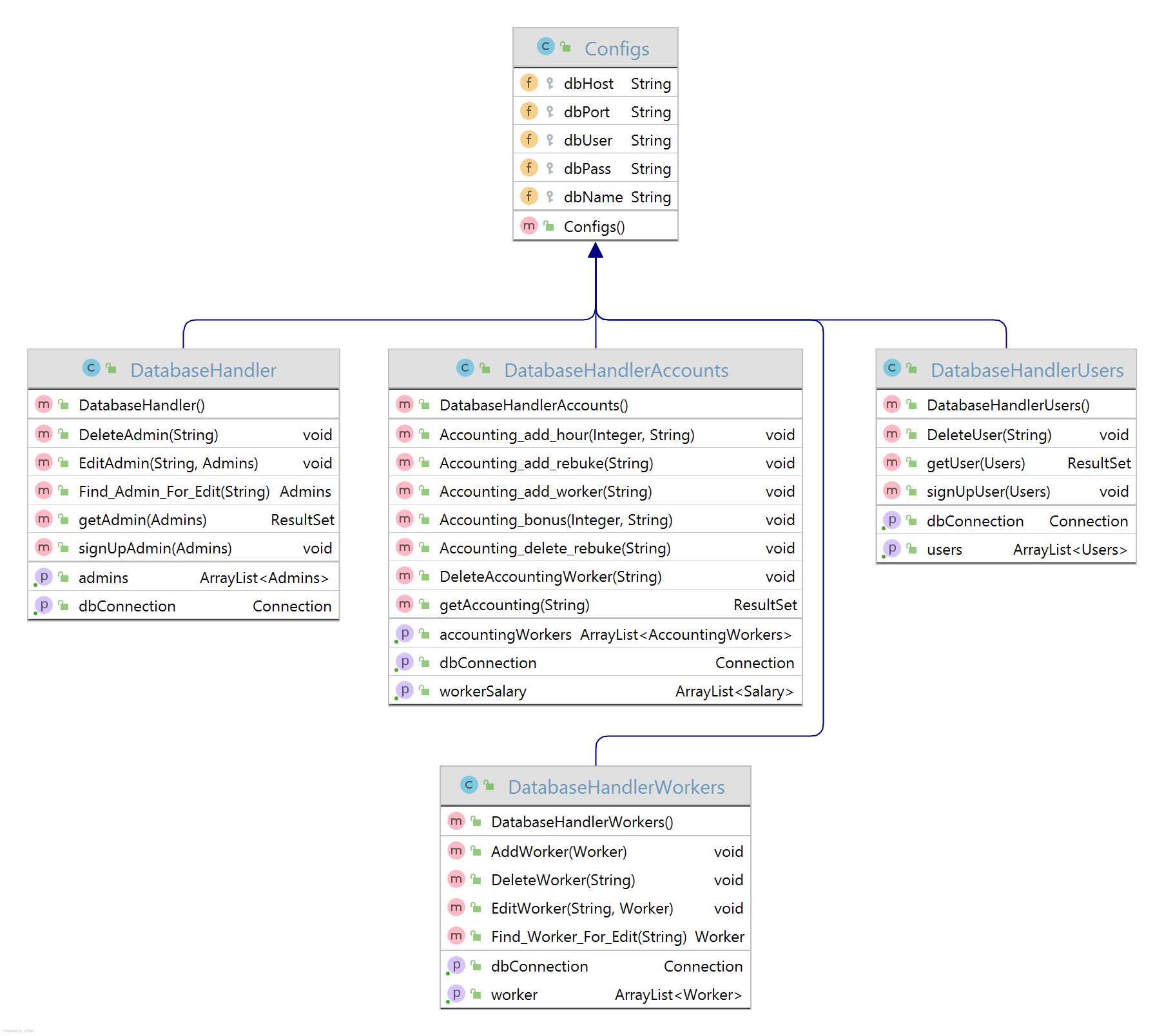


Рисунок A.3 – Классы для работы с базой данных

# Приложение Б

# (обязательное)

# Функциональные модели процессов функционирования системы учета и анализа деятельности персонала на предприятии

# Приложение В

# (обязательное)

# Листинг кода

*Файл DatabaseHandlerAccounts.java*

package com.example.courseworkc.Database;

import com.example.courseworkc.AdminServer.Configs;

import com.example.courseworkc.classes.Accounting;

import com.example.courseworkc.classes.AccountingWorkers;

import com.example.courseworkc.classes.Salary;

import com.example.courseworkc.Const.ConstAccounts;

import java.sql.\*;

import java.util.ArrayList;

public class DatabaseHandlerAccounts extends Configs {

Connection dbConnection;

public Connection getDbConnection()

throws ClassNotFoundException, SQLException {

String connectionString = "jdbc:mysql://" + dbHost + ":"

+ dbPort + "/" + dbName;

Class.forName("com.mysql.cj.jdbc.Driver");

dbConnection = DriverManager.getConnection(connectionString, dbUser, dbPass);

return dbConnection;

}

public ResultSet getAccounting(String worker\_id) {

ResultSet resSet = null;

String select = "SELECT \* FROM " + ConstAccounts.ACCOUNTS\_TABLE + " WHERE " +

ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(select);

prSt.setString(1,worker\_id);

resSet = prSt.executeQuery();

} catch (SQLException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return resSet;

}

public void Accounting\_add\_rebuke(String id){

String update = "UPDATE " + ConstAccounts.ACCOUNTS\_TABLE +

" SET " + ConstAccounts.ACCOUNTS\_REBUKE + "=?"+ " WHERE " + ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);

preparedStatement.setString(1, "есть");

preparedStatement.setString(2, id);

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public void Accounting\_delete\_rebuke(String id){

String update = "UPDATE " + ConstAccounts.ACCOUNTS\_TABLE +

" SET " + ConstAccounts.ACCOUNTS\_REBUKE + "=?"+ " WHERE " + ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);

preparedStatement.setString(1, "нет");

preparedStatement.setString(2, id);

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public void Accounting\_add\_worker(String worker\_id){

String insert = "INSERT INTO " + ConstAccounts.ACCOUNTS\_TABLE + "(" +

ConstAccounts.ACCOUNTS\_IDWORKER + "," + ConstAccounts.ACCOUNTS\_HOUR + "," +

ConstAccounts.ACCOUNTS\_BONUS + "," + ConstAccounts.ACCOUNTS\_REBUKE + ")" +

"VALUES(?,?,?,?)";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(insert);

prSt.setString(1, worker\_id);

prSt.setString(2, "0");

prSt.setString(3, "0");

prSt.setString(4, "нет");

prSt.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public void Accounting\_bonus(Integer bonus, String id){

String update = "UPDATE " + ConstAccounts.ACCOUNTS\_TABLE +

" SET " + ConstAccounts.ACCOUNTS\_BONUS + "=?"+ " WHERE " + ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);

preparedStatement.setInt(1, bonus);

preparedStatement.setString(2, id);

preparedStatement.executeUpdate();

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

}

public void Accounting\_add\_hour(Integer hour, String id){

// сначало достаем значение из таблицы

Integer HOUR;

String select = "SELECT " +"hour"+ " FROM " + ConstAccounts.ACCOUNTS\_TABLE + " WHERE " +

ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(select);

prSt.setString(1,id);

ResultSet resSet = prSt.executeQuery();

while (resSet.next()){

Accounting accounting = new Accounting();

accounting.setAcc\_hour(resSet.getString("hour"));

HOUR = Integer.parseInt(accounting.getAcc\_hour()) +hour;

System.out.println(HOUR);

String update = "UPDATE " + ConstAccounts.ACCOUNTS\_TABLE +

" SET " + ConstAccounts.ACCOUNTS\_HOUR + "=?"+ " WHERE " + ConstAccounts.ACCOUNTS\_IDWORKER + "=?";

try {

PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);

preparedStatement.setInt(1, HOUR);

preparedStatement.setString(2, id);

preparedStatement.executeUpdate();

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

}

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

}

public ArrayList<Salary> getWorkerSalary(){

ArrayList<Salary> salary = new ArrayList<Salary>();

String select ="SELECT workers.id, workers.name, workers.surname, workers.department, workers.salary, accounts.hour, accounts.bonus,accounts.rebuke FROM coursework.workers JOIN accounts ON accounts.worker\_id=workers.id;";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(select);

ResultSet resSet = prSt.executeQuery();

while (resSet.next()) {

Salary sw = new Salary();

sw.setWorker\_id(resSet.getString("id"));

sw.setWorker\_name(resSet.getString("name"));

sw.setWorker\_surname(resSet.getString("surname"));

sw.setWorker\_department(resSet.getString("department"));

sw.setWorker\_salary(resSet.getString("salary"));

sw.setAcc\_hour(resSet.getString("hour"));

sw.setAcc\_bonus(resSet.getString("bonus"));

sw.setAcc\_rebuke(resSet.getString("rebuke"));

salary.add(sw);

}

} catch (SQLException | ClassNotFoundException throwables) {

throwables.printStackTrace();

}

return salary;

}

public void DeleteAccountingWorker(String id){

String delete = "DELETE FROM " + ConstAccounts.ACCOUNTS\_TABLE + " WHERE " + ConstAccounts.ACCOUNTS\_IDWORKER + "='" + id + "'; ";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(delete);

prSt.executeUpdate();

System.out.println("id принятое на сервер = " + id);

System.out.println("Работник успешно удален из учета");

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

}

public ArrayList<AccountingWorkers> getAccountingWorkers() {

ArrayList<AccountingWorkers> workers = new ArrayList<AccountingWorkers>();

String select = "SELECT workers.name, workers.surname, workers.department, workers.position, workers.salary, accounts.hour, accounts.bonus,accounts.rebuke FROM coursework.workers JOIN accounts ON accounts.worker\_id=workers.id;";

try {

PreparedStatement prSt = getDbConnection().prepareStatement(select);

ResultSet resSet = prSt.executeQuery();

while (resSet.next()) {

AccountingWorkers aw = new AccountingWorkers();

aw.setWorker\_name(resSet.getString("name"));

aw.setWorker\_surname(resSet.getString("surname"));

aw.setWorker\_department(resSet.getString("department"));

aw.setWorker\_position(resSet.getString("position"));

aw.setWorker\_salary(resSet.getString("salary"));

aw.setAcc\_hour(resSet.getString("hour"));

aw.setAcc\_bonus(resSet.getString("bonus"));

aw.setAcc\_rebuke(resSet.getString("rebuke"));

workers.add(aw);

}

} catch (SQLException | ClassNotFoundException throwables) {

throwables.printStackTrace();

}

return workers;

}

}

*Файл AdminClient.java*

package com.example.courseworkc.controller;

import com.example.courseworkc.classes.\*;

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Objects;

import org.apache.commons.codec.digest.DigestUtils;

public class AdminClient {

private String log;

String Check\_Connect(String ip, int port){

try(Socket clientSocket = new Socket(ip,port);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Server is active. You are able to connect!";

} catch (IOException e) {

log = "Unable to find server with this credentials. Please, check the out!";

}

return log;

}

int signIn(String ip, int port, String login, String password){

int status = 0;

try(Socket clientSocket = new Socket(ip,port);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

System.out.println("Connecting...");

writer.write("authorization");

writer.newLine();

writer.write(login);

writer.newLine();

password = DigestUtils.sha256Hex(password);

writer.write(password);

writer.newLine();

writer.flush();

String success = reader.readLine(); // принимаем состояние входа

status = success.equals("success")? 1 : 0;

}

catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return status;

}

void AddAdmin(String name, String surname, String login, String password,String email){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())))

{

log = "Вы подключились к серверу";

writer.write("addAdmin");

writer.newLine();

password = DigestUtils.sha256Hex(password);

// отправляем имя фамилию логин пароль и емаил на сервер

writer.write(name);writer.newLine();

writer.write(surname);writer.newLine();

writer.write(login);writer.newLine();

writer.write(password);writer.newLine();

writer.write(email);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

}

ArrayList<AccountingWorkers> showAccountingWorkers(){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Вы подключились к серверу";

writer.write("ShowAccountingWorkers");writer.newLine();

writer.flush();

ArrayList<AccountingWorkers> arrayList = new ArrayList<AccountingWorkers>();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

arrayList = (ArrayList<AccountingWorkers>) object;

return arrayList;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return null;

}

ArrayList<Salary> ShowSalaryWorkers(){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Вы подключились к серверу";

writer.write("ShowWorkersSalary");

writer.newLine();

writer.flush();

ArrayList<Salary> arrayList = new ArrayList<Salary>();

ArrayList<Salary> salaryworkers = new ArrayList<>();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

arrayList = (ArrayList<Salary>) object;

for(Salary p : arrayList){

double sal = 0;

String rebuke = p.getAcc\_rebuke();

if(Objects.equals(rebuke, "нет")){

int percent\_bonus = Integer.parseInt(p.getAcc\_bonus());

double bonus = percent\_bonus \* 0.01;

double coefficient = bonus + 1.0;

// 4\_рабочие\_недели \* (часовая\_ставка \* часы\_в\_неделю) \* коэффициент\_премии

sal = 4.0 \* Integer.parseInt(p.getWorker\_salary()) \* Integer.parseInt(p.getAcc\_hour()) \* coefficient;

p.setWorker\_salary(String.valueOf(sal));

} else {

// 4\_рабочие\_недели \* (часовая\_ставка \* часы\_в\_неделю)

sal = 4.0 \* Integer.parseInt(p.getWorker\_salary()) \* Integer.parseInt(p.getAcc\_hour());

p.setWorker\_salary(String.valueOf(sal));

}

salaryworkers.add(p);

}

return salaryworkers;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return null;

}

ArrayList<Worker> showWorkers(){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Вы подключились к серверу";

writer.write("ShowWorkers");

writer.newLine();

writer.flush();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

// Object object = objectInputStream.readObject();

return (ArrayList<Worker>) objectInputStream.readObject();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return null;

}

ArrayList<User> showUsers(){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Вы подключились к серверу";

writer.write("showusers");writer.newLine();

writer.flush();

ArrayList<User> arrayList = new ArrayList<User>();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

arrayList = (ArrayList<User>) object;

return arrayList;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return null;

}

ArrayList<Admin> showAdmins(){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

log = "Вы подключились к серверу";

writer.write("showadmins");

writer.newLine();

writer.flush();

ArrayList<Admin> arrayList = new ArrayList<Admin>();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

arrayList = (ArrayList<Admin>) object;

return arrayList;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

log = "Не удалось подключится к серверу";

}

return null;

}

void DeleteWorker(String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("deleteWorker");writer.newLine();

writer.write(id);writer.newLine();writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void DeleteUser(String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("deleteUser");writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void DeleteAccountingWorker(String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("deleteAccountionWorker");writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void DeleteAdmin(String id){ // просто передаю id на сервак

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("deleteAdmin");writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

Worker FindWorker(String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("FindWorker");writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

Worker worker = new Worker();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

worker = (Worker) object;

return worker;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

Admin FindAdmin(String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("FindAdmin");writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

Admin admin = new Admin();

try {

ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());

try {

Object object = objectInputStream.readObject();

admin = (Admin) object;

return admin;

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

void EditWorker(String id, String Name, String Surname, String Fathername,

String department,String Position, String Salary, String Year)

{

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("EditWorker"); writer.newLine();

writer.write(id); writer.newLine();

writer.write(Name); writer.newLine();

writer.write(Surname); writer.newLine();

writer.write(Fathername); writer.newLine();

writer.write(department); writer.newLine();

writer.write(Position); writer.newLine();

writer.write(Salary); writer.newLine();

writer.write(Year); writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void EditAdmin(String id, String Name, String Surname, String Login, String Password, String Email){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("EditAdmin");writer.newLine();

writer.write(id);writer.newLine();

writer.write(Name);writer.newLine();

writer.write(Surname);writer.newLine();

writer.write(Login);writer.newLine();

writer.write(Password);writer.newLine();

writer.write(Email);writer.newLine();

writer.flush();

} catch (IOException e) {e.printStackTrace();}}

void Accounting\_add\_worker(Worker worker){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

String worker\_id = worker.getWorker\_id();

writer.write("Accounting\_add\_worker");writer.newLine();

writer.write(worker\_id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void Accounting\_add\_rebuke(String worker\_id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("Accounting\_add\_rebuke");writer.newLine();

writer.write(worker\_id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void Accounting\_delete\_rebuke(String worker\_id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("Accounting\_delete\_rebuke");writer.newLine();

writer.write(worker\_id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void Accounting\_bonus(Integer bonus,String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("Accounting\_bonus");writer.newLine();

writer.write(bonus.toString());writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void Accounting\_add\_hour(Integer hour, String id){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("Accounting\_hour");writer.newLine();

writer.write(hour.toString());writer.newLine();

writer.write(id);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

void AddWorker(String name, String surname, String fathername, String department, String position,String salary,String year){

try(Socket clientSocket = new Socket("127.0.0.1",8081);

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))

{

writer.write("AddWorker");writer.newLine();

writer.write(name);writer.newLine();

writer.write(surname);writer.newLine();

writer.write(fathername);writer.newLine();

writer.write(department);writer.newLine();

writer.write(position);writer.newLine();

writer.write(salary);writer.newLine();

writer.write(year);writer.newLine();

writer.flush();

} catch (IOException e) {

e.printStackTrace();}}}

# Приложение Г

# (обязательное)

# Скрипт генерации базы данных

CREATE TABLE `admins` (

`id` int NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL,

`lastname` varchar(45) NOT NULL,

`login` varchar(45) NOT NULL,

`password` varchar(64) NOT NULL,

`email` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `accounts` (

`id` int NOT NULL AUTO\_INCREMENT,

`worker\_id` varchar(45) NOT NULL,

`hour` int DEFAULT NULL,

`bonus` int DEFAULT NULL,

`rebuke` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id`)

) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `user` (

`id` int NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL,

`lastname` varchar(45) NOT NULL,

`login` varchar(45) NOT NULL,

`password` varchar(64) NOT NULL,

`email` varchar(45) NOT NULL,

`gender` varchar(45) NOT NULL,

`location` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `workers` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`surname` varchar(45) NOT NULL,

`fathername` varchar(45) NOT NULL,

`department` varchar(45) NOT NULL,

`position` varchar(45) NOT NULL,

`year` varchar(45) NOT NULL,

`salary` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci