Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Инженерно-экономический факультет

Кафедра экономической информатики

Дисциплина «Средства и технологии анализа и разработки информационных систем»

Курсовая работа по теме «Разработка автоматизированной системы поддержки деятельности отдела кадров предприятия»

Выполнил: Научный руководитель

студент группы 894351 Унучек Татьяна Михайловна

Галкин Илья Викторович

Минск, 2022

**Содержание**

[Введение 4](#_Toc96899434)

[1 Описание системы поддержки деятельности отдела кадров 6](#_Toc96899435)

[1.1 Описание предметной области 6](#_Toc96899436)

[1.2 Функциональное моделирование на основе стандарта IDEF0 8](#_Toc96899437)

[1.3 Определение требований к системе с точки зрения предметной области 12](#_Toc96899438)

[2 Постановка задачи и обзор методов решения 14](#_Toc96899439)

[2.1 Постановка задачи 14](#_Toc96899440)

[2.2 Технологии для решения поставленной задачи 14](#_Toc96899441)

[3 Спецификация системы (документирование вариантов использования) 18](#_Toc96899442)

[4 Модели представления системы 21](#_Toc96899443)

[4.1 Диаграммы классов системы 21](#_Toc96899444)

[4.2 Диаграмма последовательности 24](#_Toc96899445)

[4.3 Диаграмма состояния основных объектов системы 26](#_Toc96899446)

[4.4 Диаграмма компонентов системы 27](#_Toc96899447)

[4.5 Диаграмма компонентов системы 28](#_Toc96899448)

[5 Построение информационной модели системы 29](#_Toc96899449)

[6 Описание алгоритмов, реализующий бизнес-логику серверной части проектируемой системы 31](#_Toc96899450)

[6.1 Алгоритм процесса аутентификации 31](#_Toc96899451)

[6.2 Алгоритм добавления нового сотрудника 32](#_Toc96899452)

[7 Руководство пользователя 34](#_Toc96899453)

[7.1 Руководство по разворачиванию проекта 34](#_Toc96899454)

[7.2 Руководство по использованию проекта 35](#_Toc96899455)

[8 Результаты тестирования разработанной системы 38](#_Toc96899456)

[8.1 Тестирование модуля входа в систему 38](#_Toc96899457)

[8.2 Тестирование модуля добавления сотрудника 39](#_Toc96899458)

[Заключение 41](#_Toc96899459)

[Список использованных источников 42](#_Toc96899460)

[Приложение А 43](#_Toc96899461)

[Приложение B 46](#_Toc96899462)

# Введение

Техника и наука постоянно развиваются, что делает возможным существенно упростить и ускорить многие привычные процессы. В настоящее время повсеместно внедряются автоматизированные технологии. Они используются во всех сферах промышленности и производства, позволяют упростить технологический процесс и работу предприятия в целом.

Автоматизация систем управления подразумевает собой комплекс программных и аппаратных мероприятий и средств, позволяющих сократить количество персонала и улучшить работу систем. Особенно активно такие технологии сейчас внедряются в сферу экономики и управления персоналом.

Автоматизированная система не является автоматической, то есть для ее реализации и нормальной работы требуется человеческое участие.

Обычно человек-оператор выполняет основные функции управления, которые не поддаются влиянию машин. Первые автоматизированные системы появились еще в 60-е годы прошлого века, но только теперь началось их активное внедрение.

Главным предназначением АСУ является повышение производительности объекта, рост эффективности его управления, а также совершенствование методов планирования процессов управления.

АСУ активно применяются в самых разных сферах жизни и современной промышленности. В частности, они используются в системах освещения, дорожного движения, в системах информации и во всех сферах промышленного хозяйства.

Основной целью применения и использования АСУ выступает повышение эффективности и использования возможностей каждого объекта. Такие системы позволяют быстро и эффективно проводить анализ работы объекта, на основе полученных данных специалисты могут принять определенные решения и наладить производственный процесс.

Кроме того, такие автоматизированные системы существенно ускоряют выполнение сбора и обработки данных, собранных с объекта, что позволяет снизить количество решений, принимаемых человеком.

Использование АСУ повышает уровень дисциплины и уровень контроля, так как теперь осуществлять контроль над проведением работ значительно проще и удобнее.

Автоматизированные системы повышают скорость управления, снижают затраты на выполнение многих вспомогательных операций. Самым важным последствием использования АСУ является увеличение производительности, снижение затрат и потерь в процессе производства.

Внедрение таких технологий оказывает положительное влияние на состояние отечественной промышленности и экономики, а также существенно упрощает жизнь персонала.

Однако технологии требуют финансовых вложений, причем на первых этапах деньги довольно большие, ведь наличие АСУ подразумевает смену оборудования и машин. С течением времени внедрение таких технологий окупается, а их наличие дает развитие отечественному производству. [1]

Целью данной курсовой работы является создание системы, позволяющей автоматизировать процесс управления персоналом. Это позволит сотрудникам отдела кадров повысить скорость и эффективность своей работы.

Для реализации полноценной системы поддержки деятельности отдела кадров предприятия необходимо изучить основные обязанности и функции кадрового дела.

# 1 Описание системы поддержки деятельности отдела кадров

## **Описание предметной области**

Предметной областью автоматизации является учет кадров предприятия.

Функции отдела кадров ограничены следующими локальными задачами:

* оперативный контроль и учёт кадров на предприятии;
* выдача справок и выходных документов о сотрудниках, налоговых льготах и других сведений;
* анализ и обработка входной информации о персонале компании;

Отдел управления персоналом занимается персональным учетом работников, который включает прием и выбытие. Основные задачи кадрового учета, следующие:

* планирование профессиональной подготовки рабочих и специалистов (обучение, переподготовка);
* планирование повышения квалификации (обучение и переобучение) руководящих кадров;
* рациональная расстановка кадров в соответствии с уровнем их квалификации;
* аттестация и рационализация рабочих мест;
* статистический и оперативный учет и отчетность;
* анализ движения кадров;
* анализ качественного состава кадров;
* ведение архива.

Отдел управления персоналом ведёт учёт персонального состава работников, движение кадров, обучение и переобучение кадров. Также функцией отдела является составление должностных инструкций, формирование отчетности и обработка анкет. Отдел управления персоналом также обязан следить за трудовой дисциплиной, а также за продвижением по служебной лестнице сотрудников.

Отдел кадров наряду с расчетом численности активной части трудовых ресурсов фирмы, ведет расчет численности управленческого персонала. Базируясь на представленных показателях плановой численности, кадровая служба фирмы планирует прием и увольнение сотрудников, а также распределение и регулирование принятых рабочих и служащих по рабочим местам. В процессе изготовления продукции рабочие получают плановые задания, фактические результаты выполнения которых фиксируются либо в бумажных документах, либо на машинных носителях и передаются в бухгалтерию. Бухгалтерия в свою очередь, обрабатывает полученную информацию и вычисляет показатели оценки трудовой деятельности в денежном эквиваленте. До начала расчетов отдел кадров информирует бухгалтерию о текущих изменениях относительно отдельных работников (данные о пропущенном рабочем времени по причине временной нетрудоспособности о внутреннем и внешнем движении кадров, о нарушениях трудовой дисциплины и т.д.). Отдельные показатели такого рода передаются в отдел кадров, который использует их впоследствии при решении своих задач.

Специалист по кадрам должен выполнять следующие функции:

* подбор кадров;
* анализ резюме;
* собеседование;
* отбор претендентов на вакантные рабочие места;

Специалист по кадрам, осуществляет организацию кадрового учета на основе установленных правил его ведения, обязан обеспечить:

– планирование и осуществление приема и увольнения работников;

– распределение и регулирование принятых рабочих и служащих по рабочим местам;

– расчет численности активной части трудовых ресурсов и управленческого персонала;

В результате проектного исследования установлено, что отдел кадров нуждается в автоматизации своей деятельности с целью оперативной обработки данных о сотрудниках, что упрощает доступ к информации по иждивенцам, налоговым льготам, а также других данных о личном составе по бухгалтерии, на основе которых составляются таблицы с соответствующей информацией. [2]

## **Функциональное моделирование на основе стандарта IDEF0**

Построим функциональную модель для описания работы программы. Систему представляют, как взаимодействие отдельных функций и работ. Наиболее наглядно и понятно функциональная модель реализуется с помощью CASE-средства BP-win, при помощи нотаций DFD или IDEF0. При этом разрабатываются контекстная диаграмма и диаграммы детализации процессов.

Контекстная диаграмма представляет собой схему автоматизированной системы поддержки деятельности отдела кадров предприятия. Входными данными является информация о сотруднике, офисе, должности и отделе. Результатом функционирования являются общая информация о структуре предприятия и получение истории работы сотрудника. Управление осуществляется на основании трудового законодательства и нормативов входных данных. Механизмом исполнения работы являются сотрудник отдела кадров (в соответствии с рисунком 1.1).

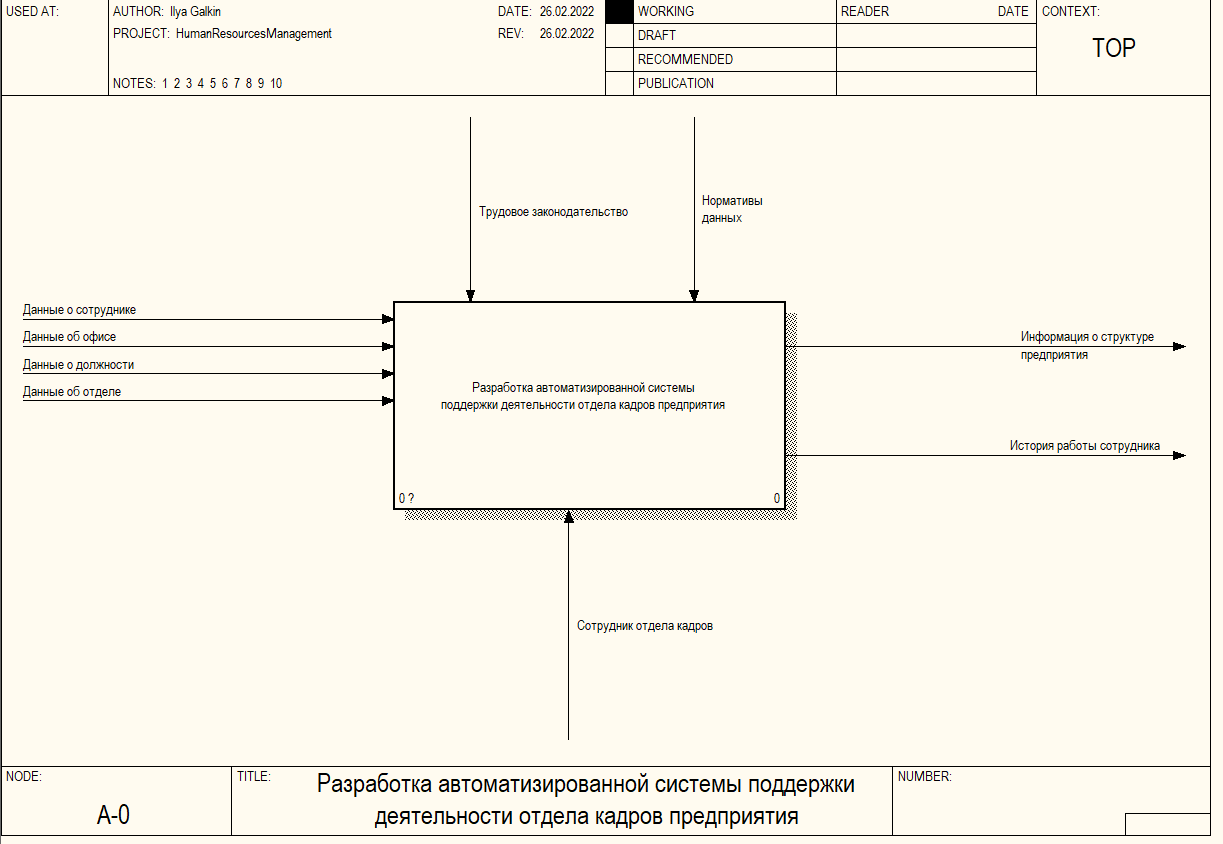


Рисунок 1.1 – Бизнес-процесс деятельности отдела кадров предприятия

В дальнейшем диаграмма детализируется на уровни. Первый уровень – это диаграмма (представлена на рисунке 1.2), с помощью которой производится детализация основного процесса, находящегося на контекстной диаграмме. Контекстную диаграмму разбиваем на 5 процессов:

1. Регистрация офиса
2. Регистрация отдела
3. Регистрация должности
4. Регистрация сотрудника
5. Назначение сотрудника

Любой блок, связанный с управлением сотрудником, находится под управлением трудового законодательства. Также, любая заполняемая форма на регистрацию управляется нормативами данных.

Для полноценного функционирования системы необходимо, в первую очередь, заполнить информацию об офисе. Такое условие обусловлено тем, что регистрация отдела невозможна без выбора офиса. И также, в случае регистрации сотрудника, необходимо заранее зарегистрировать должность.

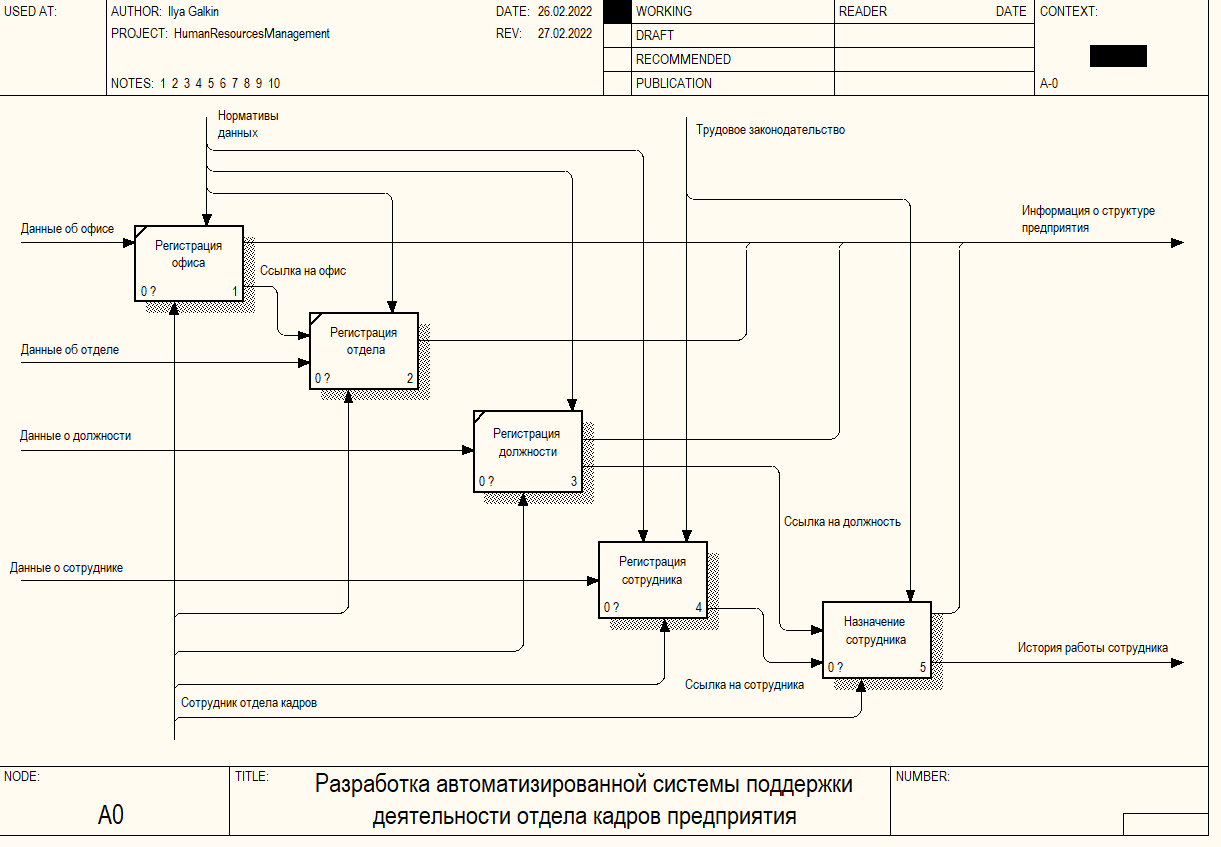


Рисунок 1.2 – Декомпозиция диаграммы верхнего уровня

На рисунке 1.3 представлена декомпозиция процесса «Регистрация сотрудника». Диаграмма состоит из следующих трех процессов:

1. Внесение данных о сотруднике
2. Валидация данных
3. Сохранение данных в базу данных

Для эффективного использования системы, сотруднику отдела кадров необходимо заранее подготовить валидные данные для добавления в базу данных подходящие под нормативы данных и трудовое законодательство.

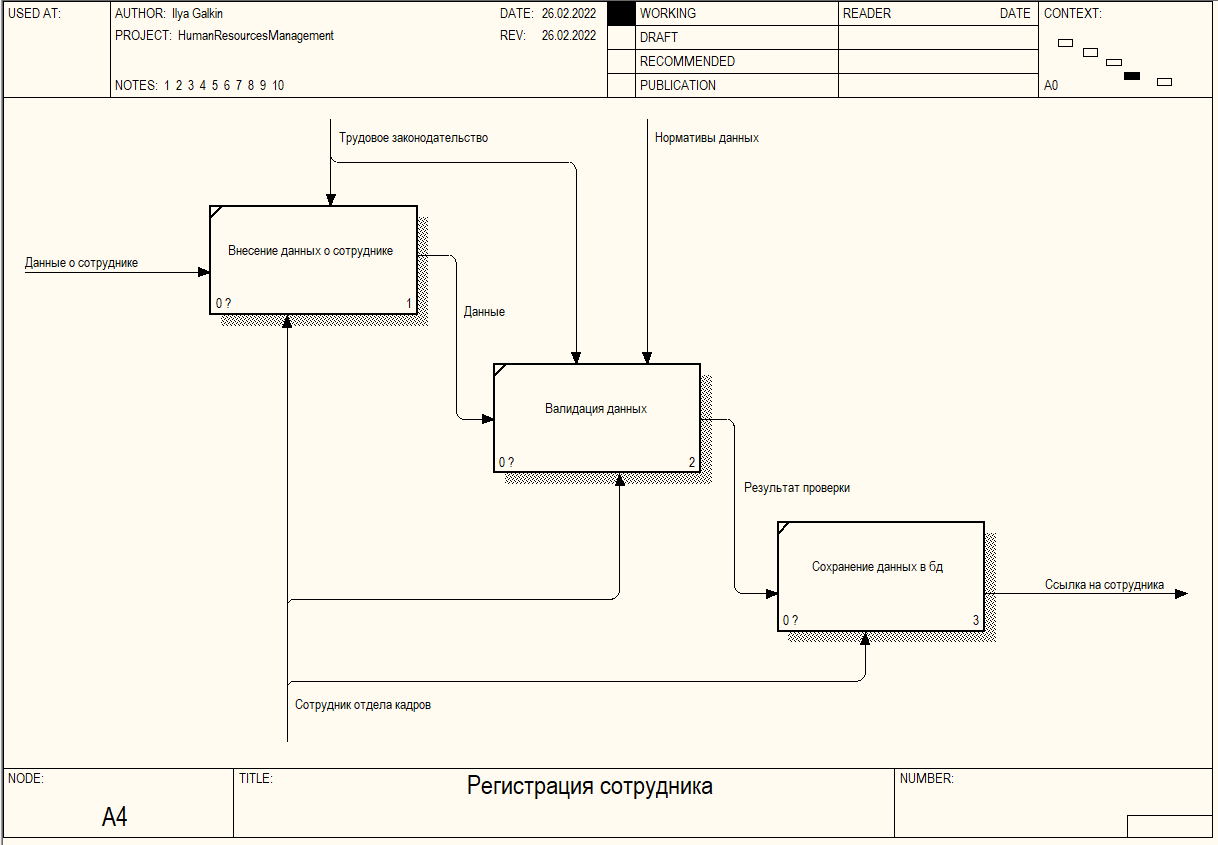


Рисунок 1.3 – Декомпозиция процесса «Регистрация сотрудника»

На рисунке 1.4 представлена декомпозиция процесса «Назначение сотрудника». Данный блок разбивается на 3 подпроцесса:

1. Выбор сотрудника и должности
2. Проверка требований к должности
3. Сохранение истории назначения

При назначении сотрудника на новую должность необходимо учитывать трудовое законодательство, нормативы данных, а также минимальную и максимальную допустимую заработную плату для указанной должности. Система не должна позволять выбор должности с некорректными данными. Также, сотрудник должен быть всегда назначен на определенную должность, что делает регистрацию должности в системе приоритетнее сотрудника.

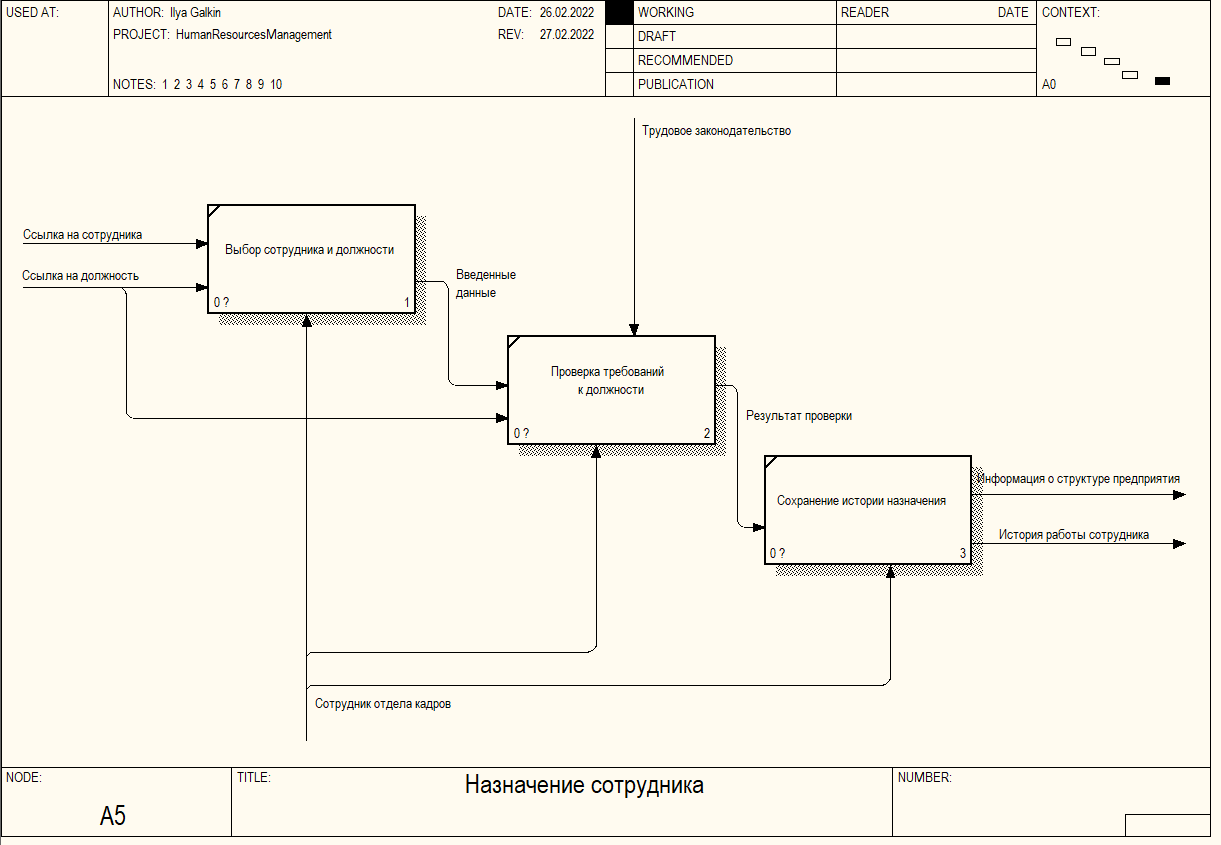


Рисунок 1.4 – Декомпозиция процесса «Назначение сотрудника»

## **Определение требований к системе с точки зрения предметной области**

Разрабатываемая система должна покрывать все основные аспекты работы сотрудника отдела кадров.

В первую очередь, необходимо реализовать эффективный и удобный для использования механизм управления сотрудниками предприятия. Данный механизм является ключевым и основополагающим для дальнейшей разработки системы.

Далее, необходимо реализовать необходимые второстепенные модули управления, которые косвенно относятся к управлению работниками. К ним относятся

1. Управление должностями
2. Управление отделами предприятия
3. Управление офисами

Все вышеизложенные модули управления дополняют информацию о сотруднике, что позволяет вести более гибкое и наглядное управление персоналом.

Также, необходимо реализовать механизм авторизации пользователей для защиты от несанкционированного доступа к системе. Причем, следует учитывать, что не каждому пользователю должно быть доступно управлением содержанием системы.

# 2 Постановка задачи и обзор методов решения

## **2.1 Постановка задачи**

По итогам работы должно быть разработано приложение, позволяющее автоматизировать деятельность сотрудника отдела кадров предприятия. Приложение должно быть интуитивно понятным для пользователя и безопасным для использования, а также должно покрывать все основные моменты работы кадрового отдела.

Приложение должно быть основано на архитектуре клиент-сервер, поддерживать механизмы аутентификации и авторизации и поддерживать логирование результатов работы.

В приложении должен быть реализован механизм обработки исключительных ситуаций и некорректного ввода, с выводом соответствующих ошибкам информационных сообщений.

Ролевая модель в приложении должна быть представлена двумя типами:

1. Пользователь
2. Администратор

Пользователю с ролью администратор должны быть доступны все функции обычного пользователя.

## **2.2 Технологии для решения поставленной задачи**

Разрабатываемая система должна быть реализована на языке программирования Java. Программы, написанные на данном языке, могут быть развернуты практически на любой платформе, так как язык считается кроссплатформенным.

Приложение должно быть разбито на клиентскую и серверную часть. Клиентская часть отвечает за интерфейс программы и за формирование запросов к серверной части и последующее обработку ответов сервера. В то же время, серверная часть отвечает за получение запросов клиентов и возвращение необходимого ответа. Язык Java позволяет реализовать подобную систему с использованием всех вышеизложенных сетевых функций.

Все данные приложения должны храниться в базе данных. Основная цель систем баз данных – обеспечение независимости данных. Независимость данных можно определить как иммунитет к изменениям в структуре хранения данных и в методах доступа к ним. Из преимуществ баз данных по сравнению с бумажными методами сохранения записей можно выделить следующие:

1. Компактность
2. Скорость
3. Низкие трудозатраты
4. Применимость
5. Соблюдение стандартов
6. Безопасность
7. Целостность данных
8. Возможность общего доступа

Для управления базой данных должна использоваться СУБД (система управлением базами данных). СУБД позволяет упростить работу с базами данных покрывая такие аспекты как:

1. Упрощение многозадачного и многопользовательского режимов
2. Введение системы безопасности
3. Резервное копирование
4. Транзакционный механизм
5. Контроль целостности данных
6. Поддержка стандартов
7. Масштабируемость
8. Администрирование

Информационная система должна быть реализована в виде web-приложения на языке Java c использованием технологий Servlet 3, JSP 2.2, EL 2.2, JSTL 1.2, XML/XSLT, HTML 5, CSS 3, ECMAScript 6. Архитектура приложения должна быть выполнена в архитектуре паттерна MVC. Интерфейс приложения должен быть оформлен с использованием каскадных таблиц стилей (CSS).

В приложении должна быть предусмотрена проверка данных, вводимых пользователем (на клиентской и серверной части проекта). В приложении должны быть разработаны и использованы собственные библиотеки тэгов. При разработке приложения должен быть использован механизм обработки исключительных ситуаций. При разработке пользовательского интерфейса необходимо предусмотреть элемент «меню».

Каркас приложения должен быть выполнен в архитектуре MVC. В приложении должны быть чётко определены уровни: модели данных, представления и контроллер. Модель данных должна быть представлена в виде набора классов, соответствующих правилам построения компонентов JavaBeans. В качестве контроллера должен выступать сервлет, который будет осуществлять взаимодействие между моделью и представлением.

Представление данных должно быть реализовано в виде набора JSP страниц. В приложении должны быть выделен слой доступа к данным (классы, в которых будут размещены методы, отвечающие за бизнес-логику системы). Доступ к данным. Доступ к данным должен быть выделен в отдельный слой приложения. Доступ к данным в СУБД должен осуществляться через драйвер JDBC, поставляемый производителем СУБД.

Использование интерфейса ODBC запрещено. База данных должна быть приведена к 3-ей нормальной форме и должно быть не менее 5 сущностей. Взаимодействие между серверной и клиентскими частями должно осуществляться с использованием сокетов и протокола HTTP. Функционал серверной части должен составлять не менее 15 Use Case.

Разработанная система должна обладать следующей инфраструктурой: - Исполняемые файлы должны работать в среде 32х разрядной ОС Windows 7 и выше - СУБД (на выбор) – Sybase SQL 11.0+, MS SQL Server 2008 R2+, MySQL 5.5+, PostgreSQL 9.0+, Java DB 10.x+ JDK 7/ JRE7. Сервлет-контейнер: Tomcat 7.0.x.

База данных должна генерироваться sql-скриптом. Интерфейс программы и данные должны быть только на русском языке. Приложение должно запускаться без использования интегрированных средств разработки. [3]

# 3 Спецификация системы (документирование вариантов использования)

Диаграмма вариантов использования (use case diagram) — диаграмма, на которой изображаются отношения между актерами и вариантами использования.

Диаграмма вариантов использования — это исходное концептуальное представление или концептуальная модель системы в процессе ее проектирования и разработки. Создание диаграммы вариантов использования имеет следующие цели:

* Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы
* Сформулировать общие требования к функциональному поведению проектируемой системы
* Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей
* Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями

Назначение данной диаграммы состоит в следующем: проектируемая программная система представляется в форме так называемых вариантов использования, с которыми взаимодействуют внешние сущности или актеры. При этом актером или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. Вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой и собственно выполнение вариантов использования. [4]

Разрабатываемый проект имеет диаграмму вариантов использования, представленную на рисунке 3.1.

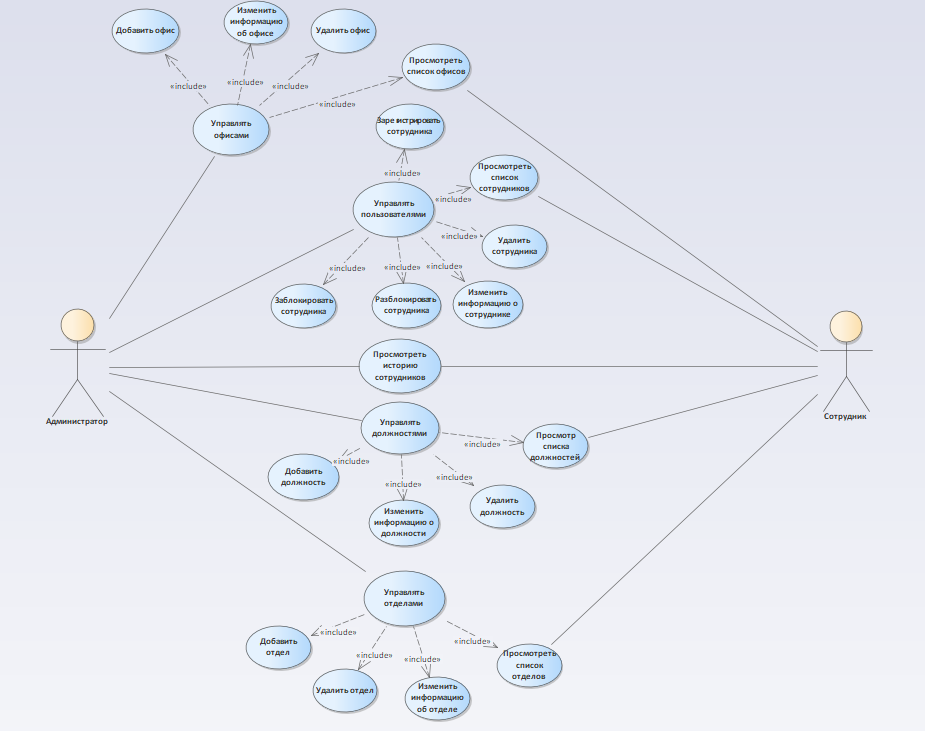


Рисунок 3.1 – Диаграмма вариантов использования

На представленной диаграмме в качестве актеров выступают администратор и сотрудник. Администратор имеет полный доступ к следующим модулям:

1. Управление офисами – создание, редактирование, удаление и чтение данных по офисам
2. Управление пользователями – создание, редактирование, удаление, чтение, блокировка и разблокировка пользователей
3. Управлению должностями – создание, редактирование, удаление и чтение данных по должностям
4. Управление отделами – создание, редактирование, удаление и чтение данных по отделам
5. Просмотр истории сотрудников – просмотр всех переходов сотрудника по должностям и отделам

Сотрудник предприятия имеет доступ ко всем вышеизложенным модулям, но только с правами на чтение.

# 4 Модели представления системы

## **4.1 Диаграммы классов системы**

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы) и типы отношений (наследование, реализация интерфейсов). На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы. На этом этапе принципиально знание ООП подхода и паттернов проектирования. [5]

Основная логика работы приложения завязана на взаимодействии между сервлетами приложения и репозиториями, которые содержать все методы доступа к таблице, связанной сущностями в базе данных.

На рисунке 4.1 представлена диаграмма классов, показывающая как основные сущности приложения связанны с соответствующими репозиториями.

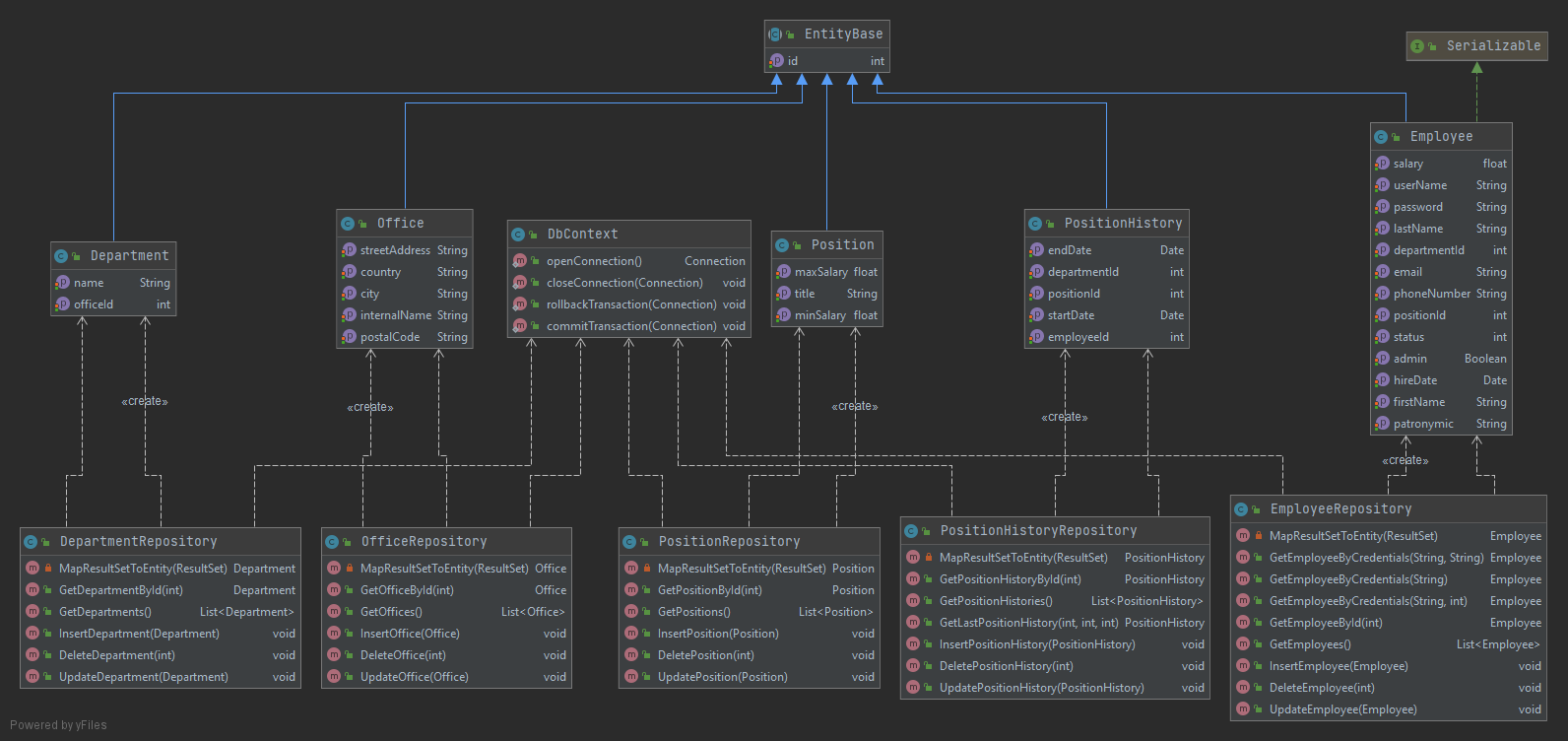


Рисунок 4.1 – Диаграмма классов репозиториев и связанных с ними сущностями

Роль связующего звена между клиентской частью приложения и серверной выполняют сервлеты. Сервлеты содержат логику по обработке всех входящих на них запросов и формированию ответа в зависимости от полученных результатов. Также, каждый из сервлетов связан со своим представлением в виде JSP-файла, что позволяет использовать полученные результаты в разметке возвращаемой страницы.

Каждый из сервлетов является агрегатором всех необходимых для работы сервисов программы. Так, каждый сервлет имеет собственный объект репозитория, позволяющий делать запросы в базу данных. Помимо этого, создается собственный объект-валидатор, проверяющий все входящие данные на сервлет и возвращающий ошибки, если таковые имеются.

Все входные данные преобразуются в объект типа ViewModel и только потом преобразуется в тип, который сохраняется в базу данных. Подобный подход позволяет разделить данные, отображающиеся на клиентской части и вносить в них изменения без обращений в базу данных.

На рисунке 4.2 представлена диаграмма классов всех используемых сервлетов приложения.



Рисунок 4.2 – Диаграмма классов сервлетов

Как уже было сказано, каждый сервлет имеет свой объект-валидатор, позволяющий проверять входные данные с клиентской части приложения. Каждый из этих объектов привязан к определенному типу ViewModel, тем самым обеспечивая надежность проверки данных.

Диаграмма классов моделей представлений и связанных с ними типами-валидаторами представлены на рисунке 4.3.

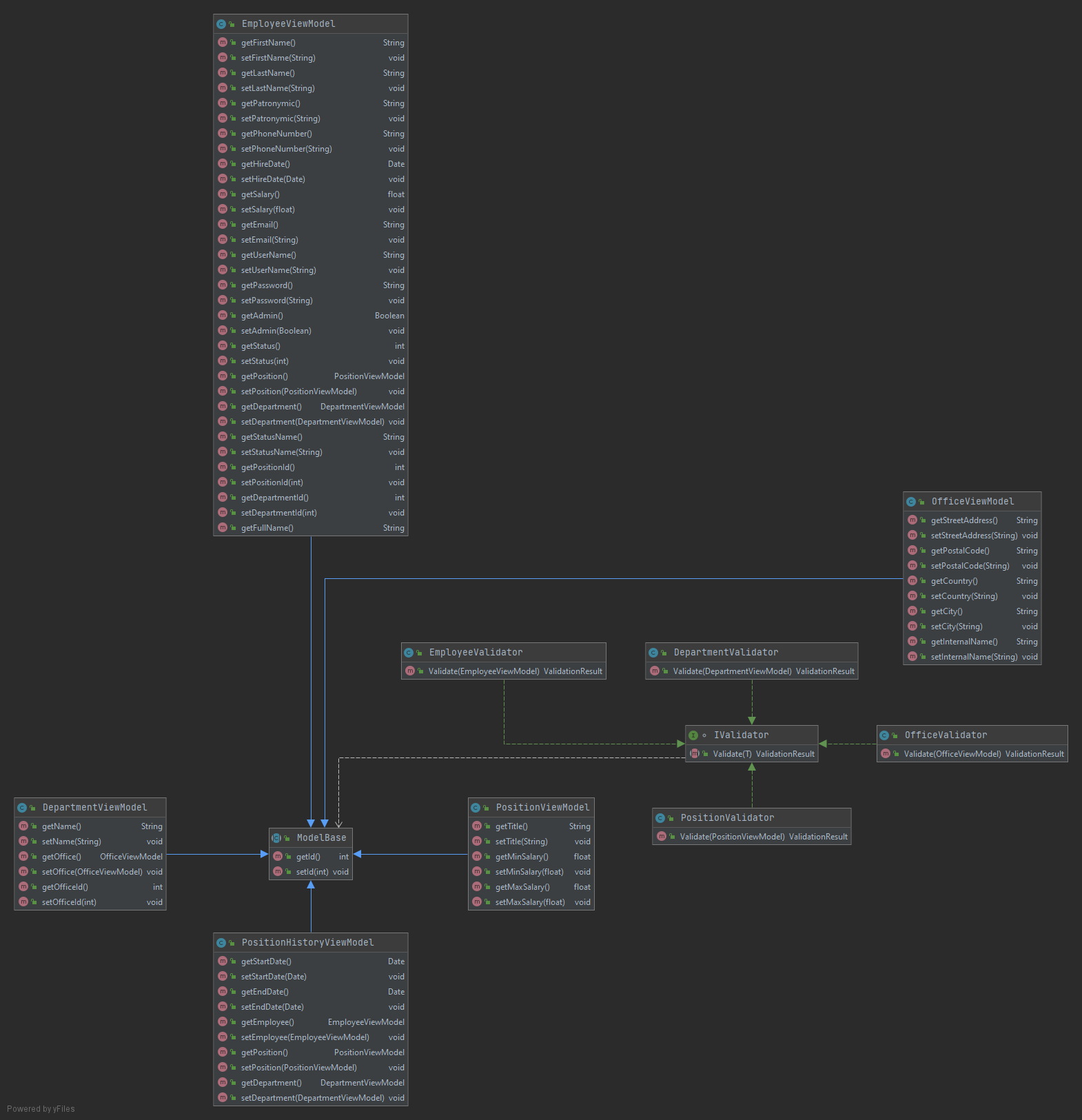


Рисунок 4.3 – Диаграмма классов моделей представления и связанных с ними типами-валидаторами

## **4.2 Диаграмма последовательности**

Диаграмма последовательности отображает взаимосвязанную работу нескольких компонентов бизнес-процесса. Они представляют собой иллюстрацию потока событий в результате использования приложения.

На рисунке 4.4 изображена диаграмма последовательности добавления нового отдела в базу данных

Для данного приложения можно выделить следующие компоненты:

1. Пользователь
2. Браузер
3. JSP-представление
4. Сервлет, принимающий запросы
5. Валидатор приходящих данных
6. Репозиторий, отвечающий за взаимодействие с базой данных
7. База данных

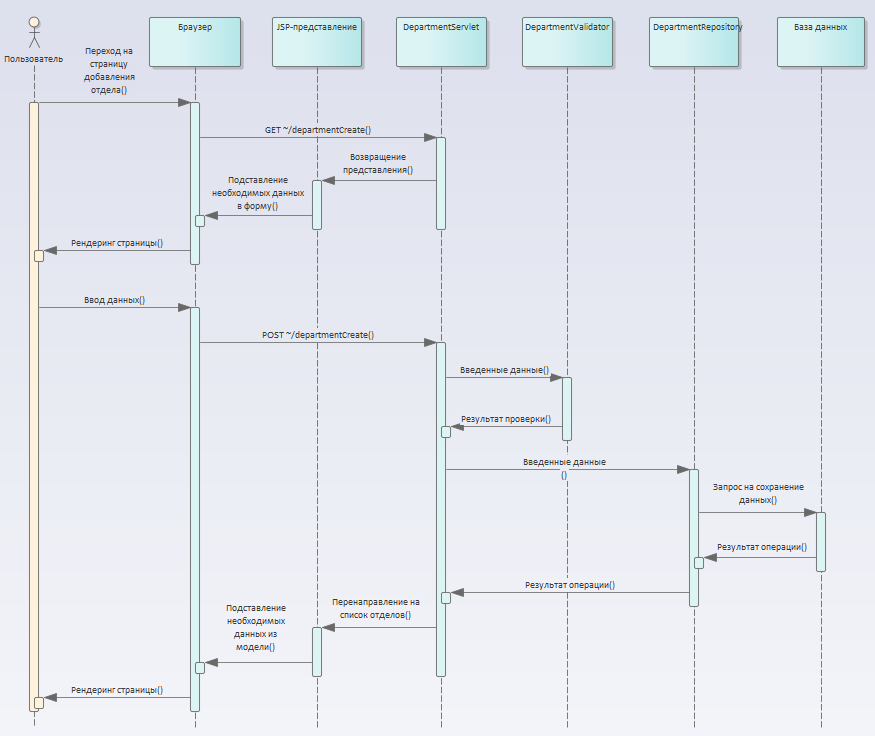


Рисунок 4.4 – Диаграмма последовательности добавления нового отдела в систему

## **4.3 Диаграмма состояния основных объектов системы**

Диаграмма состояний показывает, как объект переходит из одного состояния в другое. Очевидно, что диаграммы состояний служат для моделирования динамических аспектов системы. Диаграмма состояний полезна при моделировании жизненного цикла объекта.

От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события. [6]

Диаграмма состояния объектов изображена на рисунке 4.5.

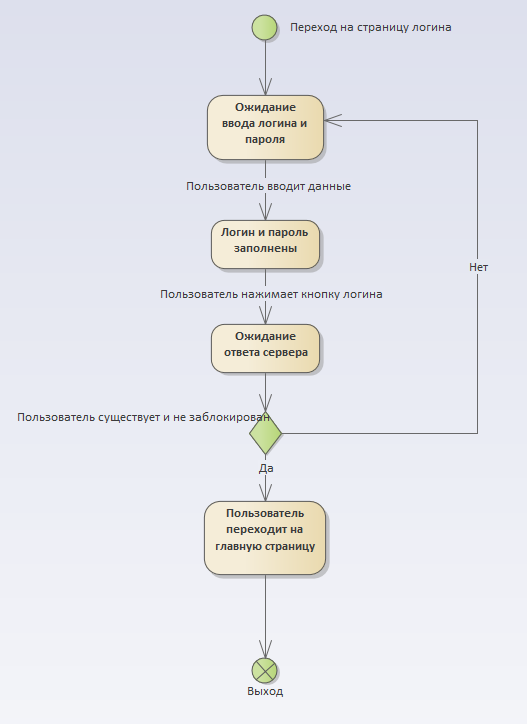


Рисунок 4.5 – Диаграмма состояния объектов

## **4.4 Диаграмма компонентов системы**

Диаграмма описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними. [7]

На рисунке 4.6 изображена диаграмма компонентов разработанной системы.

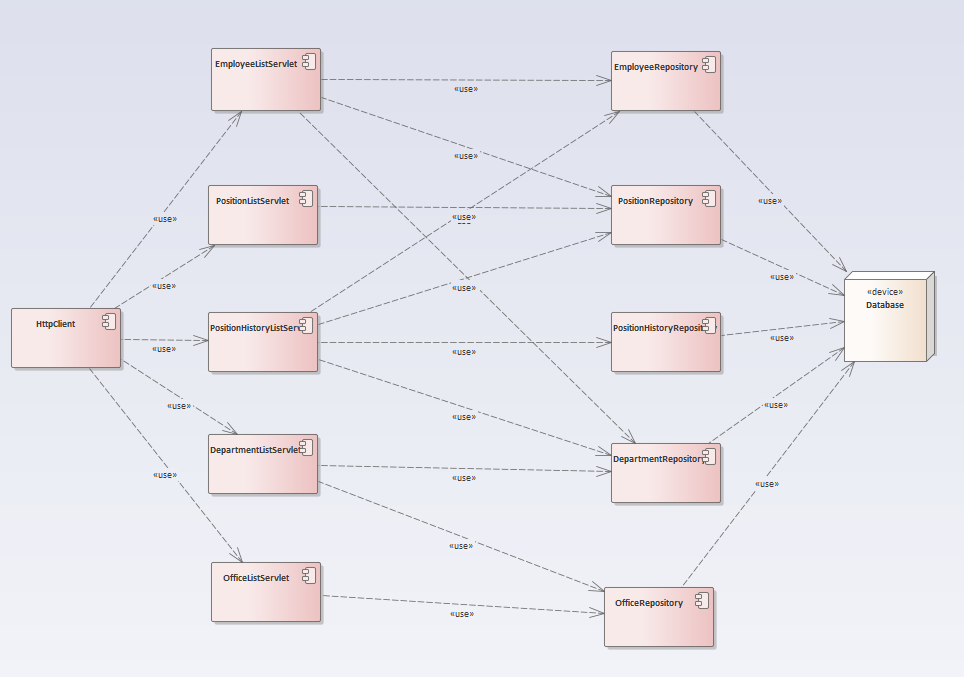


Рисунок 4.6 – Диаграмма компонентов

## **4.5 Диаграмма компонентов системы**

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. [7]

На рисунке 4.7 изображена диаграмма развертывания разработанной системы.

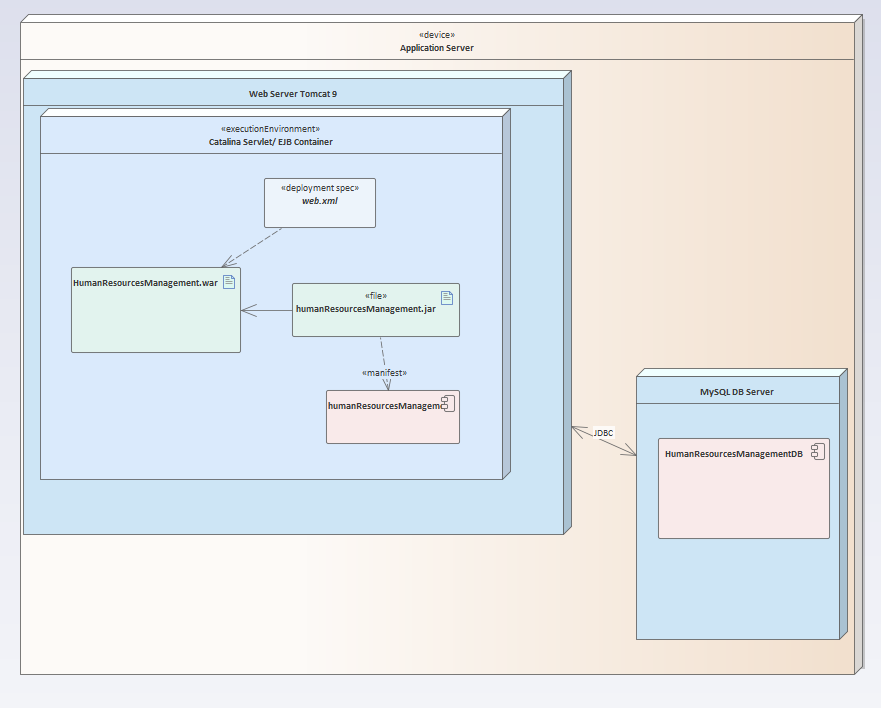


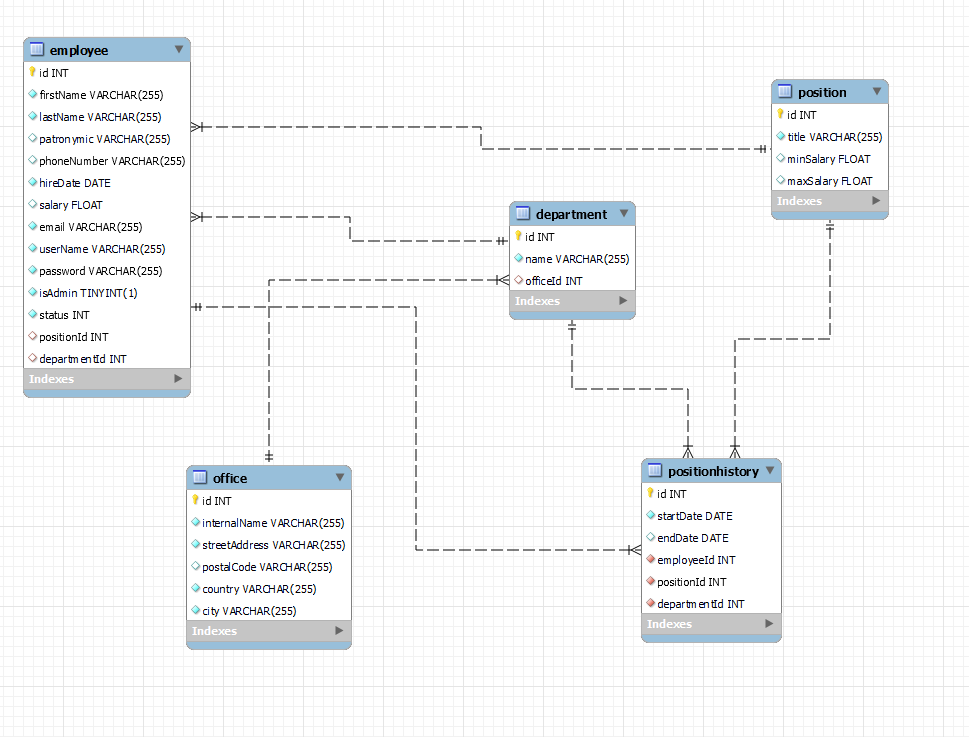
Рисунок 4.7 – Диаграмма развертывания

# 5 Построение информационной модели системы

Информационной модель разработанной системы состоит из 5 элементов:

1. Сотрудник (таблица employee)
2. Офис (таблица office)
3. Должности (таблица position)
4. Отдел (таблица department)
5. История сотрудника (таблица positionHistory)

На рисунке 4.1 представлена диаграмма информационной модели базы данных системы.

  
Рисунок 5.1 – Диаграмма информационной модели базы данных

Подобная структура подходит под требования третьей нормальной формы, так как:

1. Все атрибуты таблиц атомарны, то есть ни один из атрибутов нельзя разделить на более простые (условие 1 нормальной формы)
2. Все не ключевые атрибуты неприводимо зависимы от первичного ключа (условие 2 нормальной формы)
3. В таблицах отсутствует транзитивная зависимость, то есть не ключевые столбцы не зависят от значений других не ключевых столбцов (условие 3 нормальной формы) [8]

# 6 Описание алгоритмов, реализующий бизнес-логику серверной части проектируемой системы

## **6.1 Алгоритм процесса аутентификации**

Алгоритм начинается с ввода логина и пароля пользователем. Если пользователь оставит поля с логином и паролем пустыми, то выведется сообщение с ошибкой. Если ввод корректный, то клиентский код формирует запрос на сервер. На сервере идет проверка наличия пары логина и пароля в базе данных и, если комбинация не найдена, возвращается ответ с ошибкой. В случае, если пользователь был найден в БД, приложение проверяет статус пользователя и, если он заблокирован, то также сервер возвращает ответ с ошибкой. Если же все проверки были пройдены, пользователю возвращается представление главной страницы.

Блок-схема алгоритма представлена на рисунке 6.1

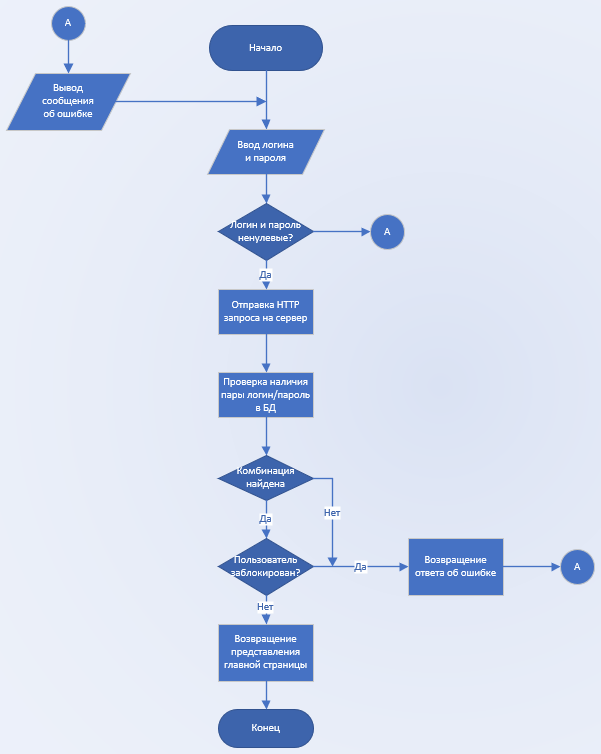


Рисунок 6.1 – Блок-схема процесса входа в систему

## **6.2 Алгоритм добавления нового сотрудника**

Алгоритм начинается с перехода на форму ввода логина и пароля. После того, как данные были введены, идет процесс аутентификации и, если пользователь не найден, то алгоритм возвращается в самое начало. Если пользователь найден, то далее идет переход на страницу главную страницу с сохранением объекта пользователя в сессию клиента.

Далее, клиент переходит на страницу управления сотрудниками. При загрузке страницы проверяется, есть ли доступ у пользователя на изменение сотрудников. Если нет, то алгоритм переходит в конец, так как кнопка «Добавить сотрудника» не отобразится в браузере. При наличии доступа кнопка будет отображаться и по нажатии, клиент перенаправляется на форму добавления сотрудника. Причем, на сервере, перед отправкой страницы, проверяется доступ пользователя. Это необходимо для предотвращения несанкционированного доступа к странице по ссылке.

Клиент заполняет данные и отправляет форму на сервер. Сервер, при получении запроса, в очередной раз проверяет доступ текущего пользователя к операции и только после этого приступает к обработке запроса.

В первую очередь, проверяется валидность введенных данных. В случае наличия ошибок, сервер возвращает страницу формы с соответствующим текстом ошибки. Если ошибок нет, то идет сохранение данных о пользователе в базу данных вместе с сохранением начальной истории пользователя.

Последним этапом является возвращение пользователя на страницу управления сотрудниками.

Данный алгоритм представлен на рисунке 6.2

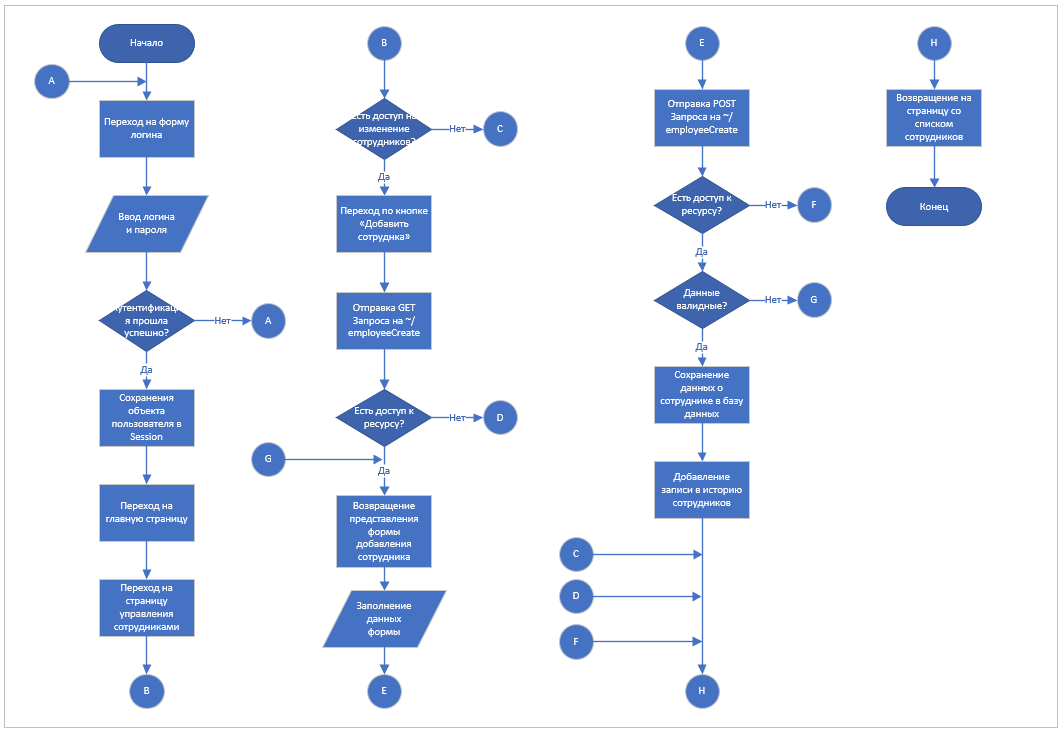


Рисунок 6.2 – Блок-схема процесса добавления нового сотрудника

# 7 Руководство пользователя

## **7.1 Руководство по разворачиванию проекта**

Для запуска приложения на локальном сервере необходимо установить заранее:

* JDK 11
* Maven
* MySQL \*
* Tomcat 9

Алгоритм разворачивания, следующий:

1. (Опционально) Откройте командную строку в папке с проектом и

сгенерируйте war файл при помощи Maven, вызвав команду mvn package

1. Создайте схему базы данных используя скрипт db\_create.sql из папки database. Затем, используйте скрипт db\_seed.sql для наполнения базы данных. В случае необходимости отката базы, используется скрипт db\_drop.sql.
2. Запустите сервер Tomcat. Это можно сделать при помощи файла bin/startup.bat из папки Tomcat.
3. Перейдите на по адресу <http://localhost:8080/manager/>. Порт может быть другим, если был изменен в конфигурационном файле. Также необходимо заранее настроить доступ к серверу в конфигурации.
4. Зайдите под пользователем admin/admin (может быть изменено в конфигурационном файле)
5. В секции WAR файл для развёртывания выберите сгенерированный на первом шаге файл, рисунок 7.1.

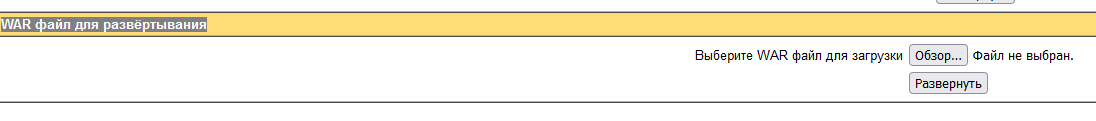


Рисунок 7.1 – Секция с загрузкой .war файла

1. После загрузки файла, в секции Приложения должно появится приложение в статусе Запущено = true, рисунок 7.2

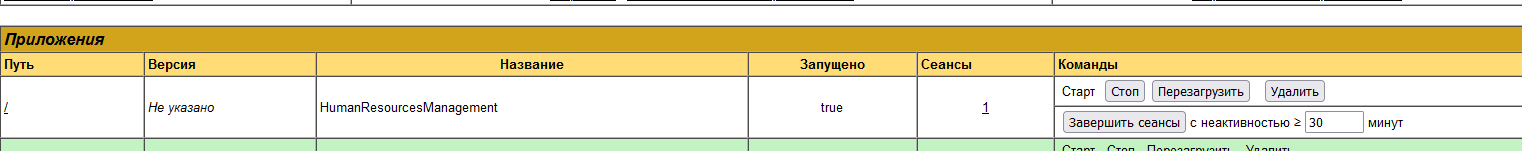
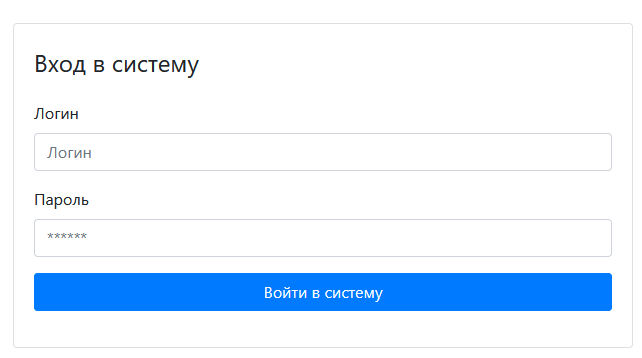


Рисунок 7.2 – Результат успешного разворачивания проекта

1. Теперь можно перейти на сайт нажав на ссылку в Путь.

## **7.2 Руководство по использованию проекта**

При переходе на сайт, пользователю открывается окно логина (рисунок 7.3)

Рисунок 7.3 – Окно входа

На данной странице пользователю необходимо ввести свой логин и пароль для входа в систему и перехода на главную страницу.

Особенность системы в том, что регистрация происходит только по инициативе существующего пользователя. Поэтому мануальная регистрация не предусмотрена.

После логина, пользователю открывается главное меню приложения (Рисунок 7.4)

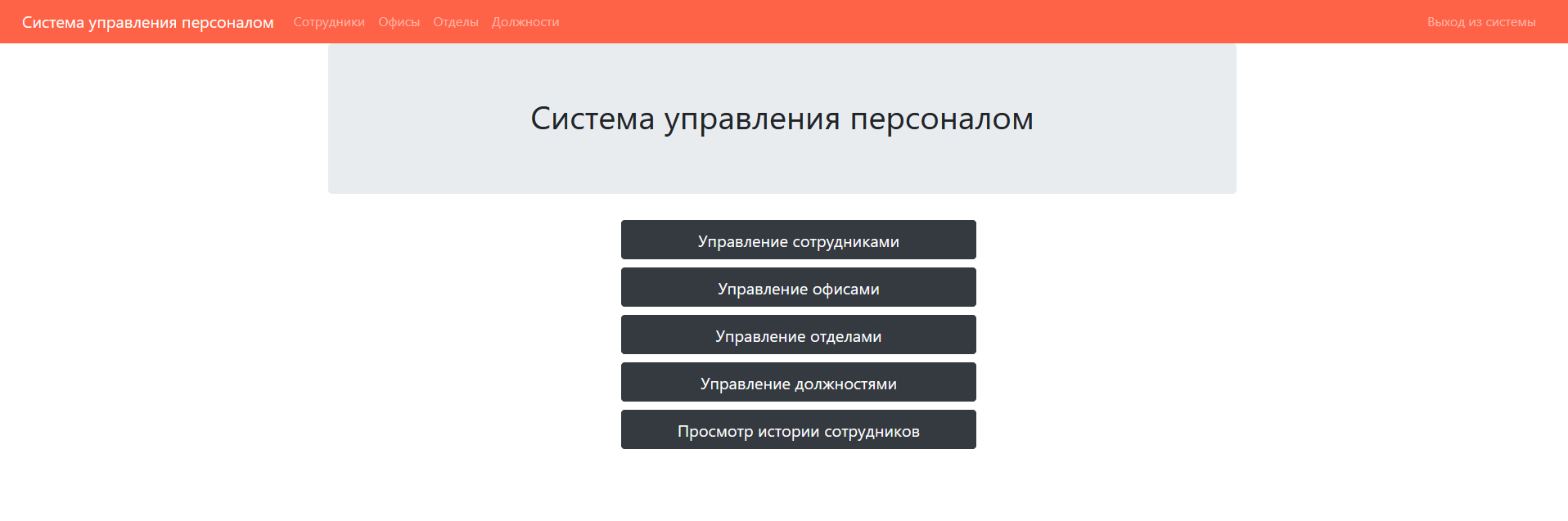


Рисунок 7.4 – Главное меню приложения

Здесь, перейдя по соответствующим кнопка в центре экрана, либо по навигационной панели сверху, пользователь может перейти на нужную ему страницу управления.

Перейдя по кнопке, пользователю представляется страница со списком элементов уже существующих в базе данных (рисунок 7.5).

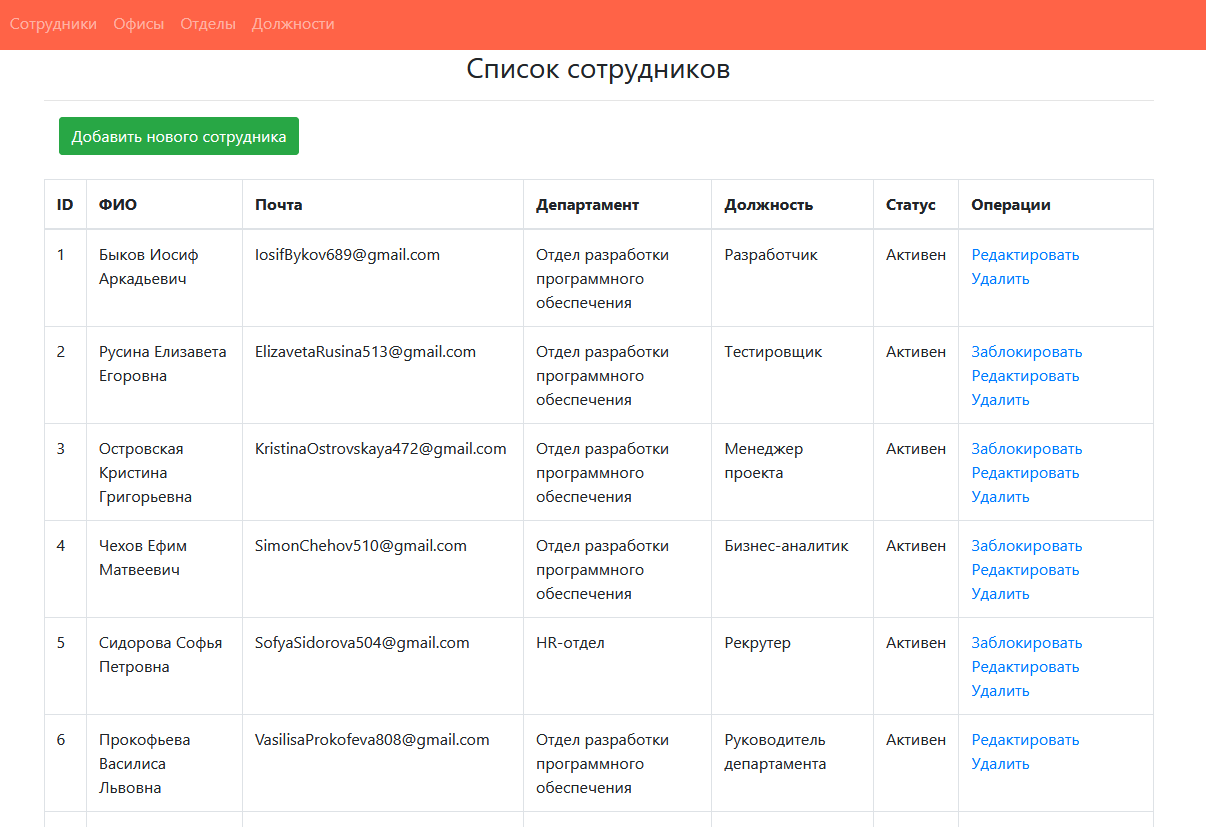


Рисунок 7.5 – Страница управления сотрудниками

На странице предусмотрено несколько возможных действий:

1. Перейти на форму добавления нового пользователя
2. Перейти на форму изменения информации уже существующего пользователя
3. Заблокировать или разблокировать пользователя
4. Удалить пользователя

Все остальные страницы управлением данных предусматривают похожую функциональность.

# Результаты тестирования разработанной системы

Разработанная система поддерживает обработку исключительных ситуаций и адекватно реагирует на некорректный ввод данных.

## **8.1 Тестирование модуля входа в систему**

При вводе несуществующих данных в окне входа, пользователю выдается соответствующее сообщение (рисунок 8.1).

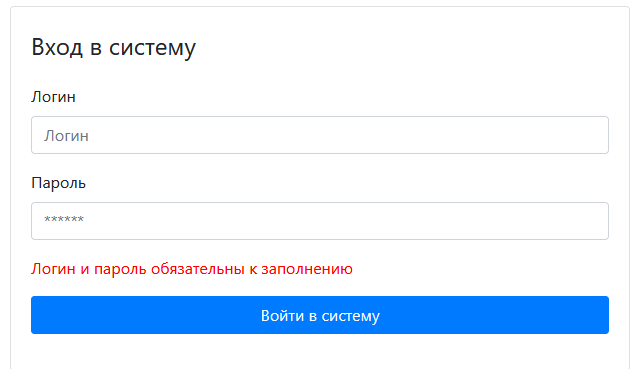


Рисунок 8.1 – Ввод несуществующего логина или пароля

При попытке входа под заблокированным пользователем, выдается информирующее сообщение (рисунок 8.2)

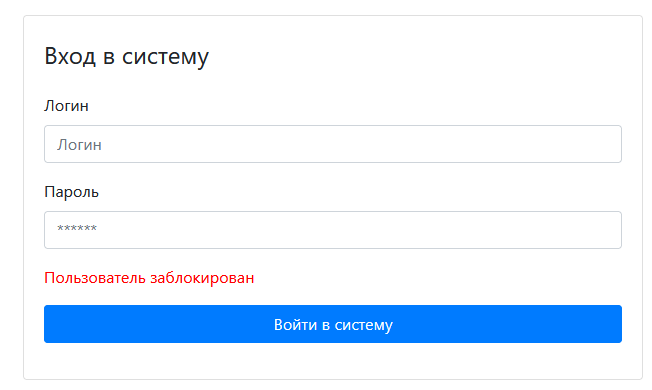


Рисунок 8.2 – Попытка входа под заблокированным пользователем

## **8.2 Тестирование модуля добавления сотрудника**

Модуль добавления сотрудника предусматривает 3 вида проверок:

1. Проверка обязательности полей (рисунок 8.3)
2. Проверка на существующий логин (рисунок 8.4)
3. Проверка, что зарплата не превышает или ниже, чем заданный диапазон на должности (рисунок 8.5)

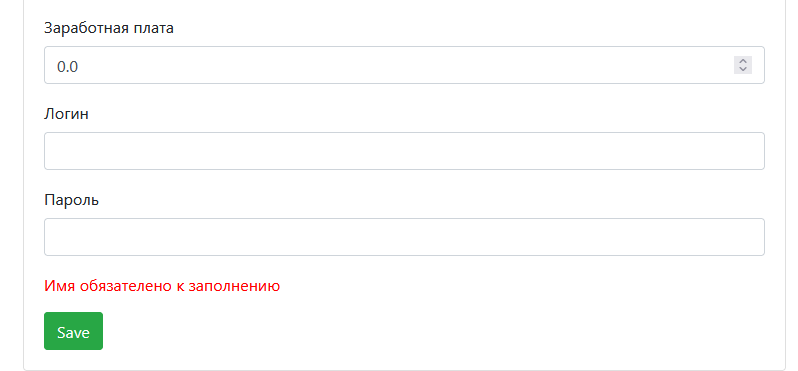


Рисунок 8.3 – Попытка добавить пользователя с незаполненным именем

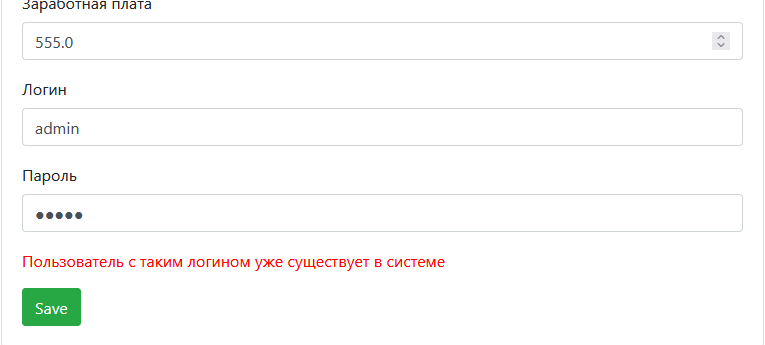


Рисунок 8.4 – Попытка добавить пользователя с уже существующим логином

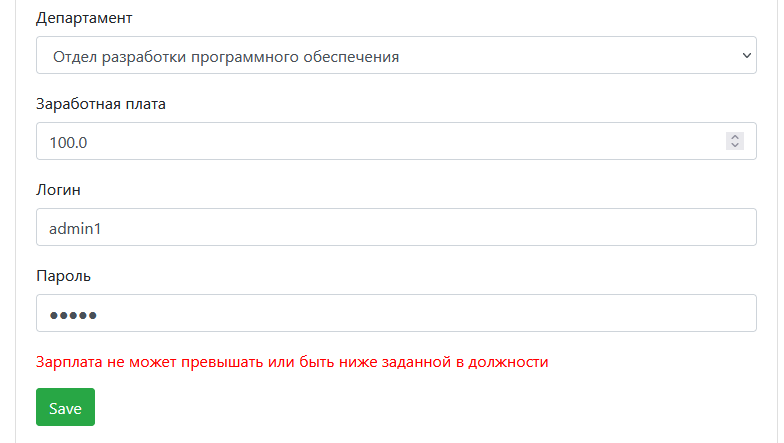


Рисунок 8.5 – Попытка добавить пользователя с зарплатой ниже заданной в должности

# Заключение

По результатам курсовой работы была создана автоматизированная система поддержки деятельности отдела кадров предприятия. Во время разработки программы были учтены все требования, предъявляемые к приложению такого рода.

Подобная система может помочь владельцам компаний повысить эффективность и скорость работы кадрового отдела предприятия.

В результате написания работы были изучены принципы разработки веб-приложения на Java с использованием технологий JSP и Servlet. Изучена база данных MySQL и были закреплены общие знания по UML-проектированию.

Данная курсовая работа является лишь первой версией разрабатываемой системы. В дальнейшем планируется добавление нескольких новых модулей, таких как: предоставление отчетных графиков, расширение базы данных приложения.

# Список использованных источников

[1] Сайт Препод24 [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://prepod24.ru/readyworks/134728/>

[2] Сайт Vuzlit [Электронный ресурс]. – Электронные данные. – Режим доступа: https://vuzlit.ru/958860/analiz\_predmetnoy\_oblasti

[3] Сайт БГУИР [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.bsuir.by/m/12_100229_1_112100.pdf>

[4] Сайт Студопедия [Электронный ресурс]. – Электронные данные. – Режим доступа:<https://studopedia.net/4_36431_otnosheniya-na-diagramme-variantov-ispolzovaniya.html>

[5] Сайт Хабр [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habr.com/en/post/74330/>

[6] Сайт ИНТУИТ [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://intuit.ru/studies/courses/1007/229/lecture/5954?page=4>

[7] Сайт Хабр [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habr.com/en/post/74330/>

[8] Сайт Заметки IT специалиста [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://info-comp.ru/third-normal-form>

# Приложение А

**(обязательное)**

**Листинг скриптов для базы данных**

db\_create.sql

**CREATE** **TABLE** IF **NOT** **EXISTS** `**position**` (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

title VARCHAR(**255**) **NOT** **NULL**,

minSalary FLOAT **NULL**,

maxSalary FLOAT **NULL**

);

**CREATE** **TABLE** IF **NOT** **EXISTS** `office` (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

internalName VARCHAR(**255**) **NOT** **NULL**,

streetAddress VARCHAR(**255**) **NOT** **NULL**,

postalCode VARCHAR(**255**) **NULL**,

country VARCHAR(**255**) **NOT** **NULL**,

city VARCHAR(**255**) **NOT** **NULL**

);

**CREATE** **TABLE** IF **NOT** **EXISTS** `department` (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

name VARCHAR(**255**) **NOT** **NULL**,

officeId INT **NULL**,

**FOREIGN** **KEY** (officeId) **REFERENCES** office(id) **ON** **DELETE** **SET** **NULL**

);

**CREATE** **TABLE** IF **NOT** **EXISTS** `employee` (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

firstName VARCHAR(**255**) **NOT** **NULL**,

lastName VARCHAR(**255**) **NOT** **NULL**,

patronymic VARCHAR(**255**) **NULL**,

phoneNumber VARCHAR(**255**) **NULL**,

hireDate DATE **NOT** **NULL**,

salary FLOAT **NULL**,

email VARCHAR(**255**) **NOT** **NULL**,

userName VARCHAR(**255**) **NOT** **NULL**,

password VARCHAR(**255**) **NOT** **NULL**,

isAdmin BOOLEAN **NOT** **NULL**,

status INT **NOT** **NULL**,

positionId INT **NULL**,

departmentId INT **NULL**,

**FOREIGN** **KEY** (positionId) **REFERENCES** `**position**`(id) **ON** **DELETE** **SET** **NULL**,

**FOREIGN** **KEY** (departmentId) **REFERENCES** department(id) **ON** **DELETE** **SET** **NULL**

);

**CREATE** **TABLE** IF **NOT** **EXISTS** `positionHistory` (

id INT AUTO\_INCREMENT **PRIMARY** **KEY**,

startDate DATE **NOT** **NULL**,

endDate DATE **NULL**,

employeeId INT **NOT** **NULL**,

positionId INT **NOT** **NULL**,

departmentId INT **NOT** **NULL**,

**FOREIGN** **KEY** (employeeId) **REFERENCES** employee(id) **ON** **DELETE** **CASCADE**,

**FOREIGN** **KEY** (positionId) **REFERENCES** `**position**`(id) **ON** **DELETE** **CASCADE**,

**FOREIGN** **KEY** (departmentId) **REFERENCES** department(id) **ON** **DELETE** **CASCADE**

);

db\_seed.sql

**INSERT** `**position**` (id, title, minSalary, maxSalary) **VALUES**

(**1**, 'Разработчик', **300**, **4000**),

(**2**, 'Тестировщик', **200**, **2000**),

(**3**, 'Менеджер проекта', **500**, **5000**),

(**4**, 'Бизнес-аналитик', **250**, **2500**),

(**5**, 'Рекрутер', **200**, **2000**),

(**6**, 'Руководитель департамента', **3000**, **10000**),

(**7**, 'Менеджер по продажам', **200**, **1500**),

(**8**, 'Юрист', **200**, **2000**),

(**9**, 'Менеджер по персоналу', **150**, **1000**),

(**10**, 'Менеджер по работе с клиентами', **200**, **1500**),

(**11**, 'Менеджер по развитию бизнеса', **400**, **3000**);

**INSERT** `office` (id, internalName, streetAddress, postalCode, country, city) **VALUES**

(**1**, 'ТВ-13', 'ул. Тверская, дом 13', '125009', 'Россия', 'Москва'),

(**2**, 'ЧГ-40', 'ул. Чигорина, дом 40, офис №3', '630066', 'Украина', 'Киев'),

(**3**, 'СР-15', 'ул. Сторожовская, дом 15', '220002', 'Республика Беларусь', 'Минск');

**INSERT** `department` (id, name, officeId) **VALUES**

(**1**, 'Отдел разработки программного обеспечения', **3**),

(**2**, 'Отдел маркетинга и PR', **2**),

(**3**, 'HR-отдел', **3**),

(**4**, 'Юридический отдел', **1**),

(**5**, 'Отдел развития', **1**);

-- id = 1, 6, 12 are admins

**INSERT** `employee` (id, firstname, lastname, patronymic, phonenumber, hireDate, salary,

email, username, password, isAdmin, status, positionId, departmentId) **VALUES**

(**1**, 'Иосиф', 'Быков', 'Аркадьевич', '+7 (955) 378-66-40', (**SELECT** **CURRENT\_DATE** - interval **200** **DAY**), **3000**, 'IosifBykov689@gmail.com', 'IosifBykov689', 'SkCr0hEvIyP8', **1**, **1**, **1**, **1**),

(**2**, 'Елизавета', 'Русина', 'Егоровна', '+7 (911) 271-74-99', (**SELECT** **CURRENT\_DATE** - interval **40** **DAY**), **1000**, 'ElizavetaRusina513@gmail.com', 'ElizavetaRusina513', 'YYayUUOUyztc', **0**, **1**, **2**, **1**),

(**3**, 'Кристина ', 'Островская ', 'Григорьевна', '+7 (959) 957-92-25', (**SELECT** **CURRENT\_DATE** - interval **25** **DAY**), **3000**, 'KristinaOstrovskaya472@gmail.com', 'KristinaOstrovskaya472', 'tPrCGjv1o3GU', **0**, **1**, **3**, **1**),

(**4**, 'Ефим', 'Чехов', 'Матвеевич', '+7 (996) 205-44-44', (**SELECT** **CURRENT\_DATE** - interval **100** **DAY**), **2000**, 'SimonChehov510@gmail.com', 'SimonChehov510', 'Pn9h2gfgmzYG', **0**, **1**, **4**, **1**),

(**5**, 'Софья', 'Сидорова', 'Петровна', '+7 (913) 865-29-56', (**SELECT** **CURRENT\_DATE** - interval **100** **DAY**), **1500**, 'SofyaSidorova504@gmail.com', 'SofyaSidorova504', '4saq1AdSnPsa', **0**, **1**, **5**, **3**),

(**6**, 'Василиса', 'Прокофьева', 'Львовна', '+7 (946) 997-22-57', (**SELECT** **CURRENT\_DATE** - interval **200** **DAY**), **5000**, 'VasilisaProkofeva808@gmail.com', 'VasilisaProkofeva808', 'kVpnkdUxkvpP', **1**, **1**, **6**, **1**),

(**7**, 'Феликс', 'Игнатьев', 'Матвеевич', '+7 (967) 351-80-09', (**SELECT** **CURRENT\_DATE** - interval **15** **DAY**), **300**, 'FeliksIgnatev963@gmail.com', 'FeliksIgnatev963', 'gOMQlgta68Y7', **0**, **1**, **7**, **5**),

(**8**, 'Ермолай', 'Царев', 'Валерьевич', '+7 (920) 530-17-30', (**SELECT** **CURRENT\_DATE** - interval **34** **DAY**), **1000**, 'ErmolayTsarev834@gmail.com', 'ErmolayTsarev834', 'wHYmmTmALw6z', **0**, **1**, **8**, **4**),

(**9**, 'Катерина', 'Чистякова ', 'Викторовна', '+7 (997) 103-85-65', (**SELECT** **CURRENT\_DATE** - interval **56** **DAY**), **500**, 'KaterinaChistyakova639@gmail.com', 'KaterinaChistyakova639', 'wHYmmTmALw6z', **0**, **1**, **9**, **3**),

(**10**, 'Ольга', 'Павлова', 'Робертовна', '+7 (992) 986-64-56', (**SELECT** **CURRENT\_DATE** - interval **23** **DAY**), **700**, 'OlgaPavlova438@gmail.com', 'OlgaPavlova438', 'ggoreVPUDgi7', **0**, **1**, **10**, **2**),

(**11**, 'Кондрат', 'Михайлов', 'Андреевич', '+7 (902) 635-97-13', (**SELECT** **CURRENT\_DATE** - interval **33** **DAY**), **3000**, 'KondratMihaylov744@gmail.com', 'KondratMihaylov744', '7Je3dioZz62A', **0**, **1**, **11**, **5**),

(**12**, 'admin', 'admin', 'admin', '+7 (777) 777-77-77', (**SELECT** **CURRENT\_DATE** - interval **1** **DAY**), **3000**, 'admin@gmail.com', 'admin', 'admin', **1**, **1**, **1**, **1**);

**INSERT** `positionHistory` (id, startDate, endDate, employeeId, positionId, departmentId) **VALUES**

(**1**,(**SELECT** **CURRENT\_DATE** - interval **200** **DAY**), **NULL**, **1**, **1**, **1**),

(**2**,(**SELECT** **CURRENT\_DATE** - interval **40** **DAY**), **NULL**, **2**, **2**, **1**),

(**3**,(**SELECT** **CURRENT\_DATE** - interval **25** **DAY**), **NULL**, **3**, **3**, **1**),

(**4**,(**SELECT** **CURRENT\_DATE** - interval **100** **DAY**), **NULL**, **4**, **4**, **1**),

(**5**,(**SELECT** **CURRENT\_DATE** - interval **100** **DAY**), **NULL**, **5**, **5**, **3**),

(**6**,(**SELECT** **CURRENT\_DATE** - interval **200** **DAY**), **NULL**, **6**, **6**, **1**),

(**7**,(**SELECT** **CURRENT\_DATE** - interval **15** **DAY**), **NULL**, **7**, **7**, **5**),

(**8**,(**SELECT** **CURRENT\_DATE** - interval **34** **DAY**), **NULL**, **8**, **8**, **4**),

(**9**,(**SELECT** **CURRENT\_DATE** - interval **56** **DAY**), **NULL**, **9**, **9**, **3**),

(**10**,(**SELECT** **CURRENT\_DATE** - interval **23** **DAY**), **NULL**, **10**, **10**, **2**),

(**11**,(**SELECT** **CURRENT\_DATE** - interval **33** **DAY**), **NULL**, **11**, **11**, **5**),

(**12**,(**SELECT** **CURRENT\_DATE** - interval **1** **DAY**), **NULL**, **12**, **1**, **1**);

db\_drop.sql

**DELETE** **FROM** `positionHistory`;

**DELETE** **FROM** `employee`;

**DELETE** **FROM** `department`;

**DELETE** **FROM** `office`;

**DELETE** **FROM** `**position**`;

**drop** **table** positionHistory

**drop** **table** employee

**drop** **table** department

**drop** **table** `**position**`

**drop** **table** office

# Приложение B

**(обязательное)**

**Листинг кода программы**

Web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<display-name>HumanResourcesManagement</display-name>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>authFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index</welcome-file>

<welcome-file>home</welcome-file>

</welcome-file-list>

<error-page>

<exception-type>java.lang.Exception</exception-type>

<location>/error.jsp</location>

</error-page>

</web-app>

Main-layout.tag

<%@tag description="layout" pageEncoding="UTF-8"%>

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="t" %>

<!DOCTYPE html>

<html>

<head>

<title>Система управления персоналом</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<style><%@include file="/WEB-INF/styles.css"%></style>

</head>

<body>

<header>

<nav class="navbar navbar-expand-md navbar-dark " style="background-color: tomato">

<div class="container-fluid">

<ul class="navbar-nav navbar-left">

<li><a href="${pageContext.request.contextPath}/" class="navbar-brand"> Система управления персоналом </a></li>

<t:if test="${user != null}">

<li><a href="${pageContext.request.contextPath}/employee" class="nav-link">Сотрудники</a></li>

<li><a href="${pageContext.request.contextPath}/office" class="nav-link">Офисы</a></li>

<li><a href="${pageContext.request.contextPath}/department" class="nav-link">Отделы</a></li>

<li><a href="${pageContext.request.contextPath}/position" class="nav-link">Должности</a></li>

</t:if>

</ul>

<t:if test="${user != null}">

<form method="post" action="logout">

<ul class="nav navbar-nav navbar-right">

<li><a onclick="this.closest('form').submit();return false;" class="nav-link" href="${pageContext.request.contextPath}/logout">Выход из системы</a></li>

</ul>

</form>

</t:if>

</div>

</nav>

</header>

<jsp:doBody/>

</body>

</html>

AuthFilter.java

**package** hrm.pipeline;

**import** **java.io.IOException**;

**import** **javax.servlet.Filter**;

**import** **javax.servlet.FilterChain**;

**import** **javax.servlet.FilterConfig**;

**import** **javax.servlet.RequestDispatcher**;

**import** **javax.servlet.ServletException**;

**import** **javax.servlet.ServletRequest**;

**import** **javax.servlet.ServletResponse**;

**import** **javax.servlet.annotation.WebFilter**;

**import** **javax.servlet.http.HttpServletRequest**;

**import** **javax.servlet.http.HttpServletResponse**;

**import** **javax.servlet.http.HttpSession**;

**@WebFilter**(filterName = "authFilter", urlPatterns = { "/\*" })

**public** **class** **AuthFilter** **implements** Filter {

**public** **void** **doFilter**(ServletRequest request, ServletResponse response, FilterChain chain)

**throws** IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

HttpServletResponse httpResponse = (HttpServletResponse) response;

HttpSession session = httpRequest.getSession(**false**);

**boolean** isLoggedIn = (session != **null** && session.getAttribute("user") != **null**);

String loginURI = httpRequest.getContextPath() + "/login";

**boolean** isLoginRequest = httpRequest.getRequestURI().equals(loginURI);

**boolean** isLoginPage = httpRequest.getRequestURI().endsWith("login.jsp");

**if** (isLoggedIn && (isLoginRequest || isLoginPage)) {

RequestDispatcher dispatcher = request.getRequestDispatcher("/");

dispatcher.forward(request, response);

} **else** **if** (isLoggedIn || isLoginRequest) {

chain.doFilter(request, response);

} **else** {

httpResponse.sendRedirect("login");

}

}

**public** **AuthFilter**() {

}

**public** **void** **destroy**() {

}

**public** **void** **init**(FilterConfig fConfig) **throws** ServletException {

}

}

DbContext.java

**package** hrm.db;

**import** **hrm.infrastructure.Constants**;

**import** **java.sql.Connection**;

**import** **java.sql.DriverManager**;

**import** **java.sql.SQLException**;

**public** **class** **DbContext** {

**public** **static** Connection **openConnection**() **throws** ClassNotFoundException, SQLException {

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

**return** DriverManager.getConnection(Constants.SQLConstants.ConnectionString,

Constants.SQLConstants.UserName,

Constants.SQLConstants.Password);

}

**public** **static** **void** **closeConnection**(Connection connection) {

**try** {

connection.close();

} **catch** (SQLException throwables) {

throwables.printStackTrace();

}

}

**public** **static** **void** **rollbackTransaction**(Connection conn) {

**try** {

conn.rollback();

} **catch** (SQLException throwables) {

throwables.printStackTrace();

}

}

**public** **static** **void** **commitTransaction**(Connection conn) {

**try** {

conn.commit();

} **catch** (SQLException throwables) {

throwables.printStackTrace();

}

}

}

EmployeeRepository.java

**package** hrm.repositories;

**import** **hrm.db.DbContext**;

**import** **hrm.entities.Employee**;

**import** **java.sql.Connection**;

**import** **java.sql.PreparedStatement**;

**import** **java.sql.ResultSet**;

**import** **java.sql.SQLException**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**public** **class** **EmployeeRepository** {

**private** Employee **MapResultSetToEntity**(ResultSet resultSet) **throws** SQLException {

Employee position = **new** Employee();

position.setId(resultSet.getInt("id"));

position.setFirstName(resultSet.getString("firstName"));

position.setLastName(resultSet.getString("lastName"));

position.setPatronymic(resultSet.getString("patronymic"));

position.setPhoneNumber(resultSet.getString("phoneNumber"));

position.setHireDate(resultSet.getDate("hireDate"));

position.setSalary(resultSet.getFloat("salary"));

position.setEmail(resultSet.getString("email"));

position.setUserName(resultSet.getString("userName"));

position.setPassword(resultSet.getString("password"));

position.setAdmin(resultSet.getBoolean("isAdmin"));

position.setStatus(resultSet.getInt("status"));

position.setPositionId(resultSet.getInt("positionId"));

position.setDepartmentId(resultSet.getInt("departmentId"));

**return** position;

}

**public** Employee **GetEmployeeByCredentials**(String username, String password) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "SELECT \* FROM Employee WHERE userName = ? && password = ?" ;

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setString(**1**, username);

sqlStatement.setString(**2**, password);

ResultSet results = sqlStatement.executeQuery();

**if**(results.next()) {

**return** **MapResultSetToEntity**(results);

}

**return** **null**;

}

**public** Employee **GetEmployeeByCredentials**(String username) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "SELECT \* FROM Employee WHERE userName = ?" ;

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setString(**1**, username);

ResultSet results = sqlStatement.executeQuery();

**if**(results.next()) {

**return** **MapResultSetToEntity**(results);

}

**return** **null**;

}

**public** Employee **GetEmployeeByCredentials**(String username, **int** excludeId) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "SELECT \* FROM Employee WHERE userName = ? && id != ?" ;

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setString(**1**, username);

sqlStatement.setInt(**2**, excludeId);

ResultSet results = sqlStatement.executeQuery();

**if**(results.next()) {

**return** **MapResultSetToEntity**(results);

}

**return** **null**;

}

**public** Employee **GetEmployeeById**(**int** id) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "SELECT \* FROM Employee WHERE Id = ?";

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setInt(**1**, id);

ResultSet results = sqlStatement.executeQuery();

**if**(results.next()) {

**return** **MapResultSetToEntity**(results);

}

**return** **null**;

}

**public** List<Employee> **GetEmployees**() **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "SELECT \* FROM Employee";

PreparedStatement sqlStatement = connection.prepareStatement(query);

ResultSet results = sqlStatement.executeQuery();

List<Employee> resultList = **new** ArrayList<>();

**while**(results.next()) {

resultList.add(MapResultSetToEntity(results));

}

**return** resultList;

}

**public** **void** **InsertEmployee**(Employee employee) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "INSERT `employee` (firstname, lastname, patronymic, phonenumber, hireDate, salary, email, username, password, isAdmin, status, positionId, departmentId) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setString(**1**, employee.getFirstName());

sqlStatement.setString(**2**, employee.getLastName());

sqlStatement.setString(**3**, employee.getPatronymic());

sqlStatement.setString(**4**, employee.getPhoneNumber());

sqlStatement.setDate(**5**, employee.getHireDate());

sqlStatement.setFloat(**6**, employee.getSalary());

sqlStatement.setString(**7**, employee.getEmail());

sqlStatement.setString(**8**, employee.getUserName());

sqlStatement.setString(**9**, employee.getPassword());

sqlStatement.setBoolean(**10**, employee.getAdmin());

sqlStatement.setInt(**11**, employee.getStatus());

sqlStatement.setInt(**12**, employee.getPositionId());

sqlStatement.setInt(**13**, employee.getDepartmentId());

sqlStatement.executeUpdate();

}

**public** **void** **DeleteEmployee**(**int** id) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "DELETE FROM Employee WHERE Id = ?";

PreparedStatement sqlStatement = connection.prepareStatement(query);

sqlStatement.setInt(**1**, id);

sqlStatement.executeUpdate();

}

**public** **void** **UpdateEmployee**(Employee employee) **throws** SQLException, ClassNotFoundException{

Connection connection = DbContext.openConnection();

String query = "UPDATE Employee SET firstname = ?, lastname = ?, patronymic = ?, phonenumber = ?, hireDate = ?, salary = ?, email = ?, username = ?, password = ?, isAdmin = ?, status = ?, positionId = ?, departmentId = ? WHERE Id = ?";

PreparedStatement sqlStatement = connection.prepareStatement(query);

Employee dbEmployee = GetEmployeeById(employee.getId());

sqlStatement.setString(**1**, employee.getFirstName());

sqlStatement.setString(**2**, employee.getLastName());

sqlStatement.setString(**3**, employee.getPatronymic());

sqlStatement.setString(**4**, employee.getPhoneNumber());

sqlStatement.setDate(**5**, dbEmployee.getHireDate());

sqlStatement.setFloat(**6**, employee.getSalary());

sqlStatement.setString(**7**, employee.getEmail());

sqlStatement.setString(**8**, employee.getUserName());

sqlStatement.setString(**9**, employee.getPassword());

sqlStatement.setBoolean(**10**, dbEmployee.getAdmin());

sqlStatement.setInt(**11**, employee.getStatus());

sqlStatement.setInt(**12**, employee.getPositionId());

sqlStatement.setInt(**13**, employee.getDepartmentId());

sqlStatement.setInt(**14**, dbEmployee.getId());

sqlStatement.executeUpdate();

}

}

EmployeeCreateServlet.java

**package** hrm.servlets;

**import** **hrm.entities.Department**;

**import** **hrm.entities.Employee**;

**import** **hrm.entities.Position**;

**import** **hrm.entities.PositionHistory**;

**import** **hrm.helpers.AuthHelper**;

**import** **hrm.helpers.DateHelper**;

**import** **hrm.infrastructure.Constants**;

**import** **hrm.infrastructure.EmployeeStatuses**;

**import** **hrm.models.EmployeeViewModel**;

**import** **hrm.models.LookupViewModel**;

**import** **hrm.models.mappers.DepartmentMapper**;

**import** **hrm.models.mappers.EmployeeMapper**;

**import** **hrm.models.mappers.PositionMapper**;

**import** **hrm.models.validators.EmployeeValidator**;

**import** **hrm.models.validators.ValidationResult**;

**import** **hrm.repositories.\***;

**import** **javax.servlet.RequestDispatcher**;

**import** **javax.servlet.ServletException**;

**import** **javax.servlet.annotation.WebServlet**;

**import** **javax.servlet.http.HttpServlet**;

**import** **javax.servlet.http.HttpServletRequest**;

**import** **javax.servlet.http.HttpServletResponse**;

**import** **java.io.IOException**;

**import** **java.sql.SQLException**;

**import** **java.util.ArrayList**;

**import** **java.util.Date**;

**import** **java.util.List**;

**@WebServlet**("/employeeCreate")

**public** **class** **EmployeeCreateServlet** **extends** HttpServlet {

**private** EmployeeRepository employeeRepository;

**private** PositionRepository positionRepository;

**private** DepartmentRepository departmentRepository;

**private** OfficeRepository officeRepository;

**private** PositionHistoryRepository positionHistoryRepository;

**private** EmployeeValidator employeeValidator;

**public** **void** **init**() {

employeeRepository = **new** EmployeeRepository();

positionRepository = **new** PositionRepository();

departmentRepository = **new** DepartmentRepository();

officeRepository = **new** OfficeRepository();

positionHistoryRepository = **new** PositionHistoryRepository();

employeeValidator = **new** EmployeeValidator();

}

**protected** **void** **doPost**(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

**if**(!AuthHelper.ValidateAdminPermission(request)) {

response.sendRedirect("/");

}

EmployeeViewModel employee = parseForm(request);

ValidationResult validationResult = **null**;

**try** {

validationResult = employeeValidator.Validate(employee);

} **catch** (SQLException | ClassNotFoundException throwables) {

request.setAttribute("errorString", throwables.getMessage());

request.setAttribute("employee", employee);

RequestDispatcher dispatcher = **this**.getServletContext().getRequestDispatcher("/employee-form.jsp");

dispatcher.forward(request, response);

**return**;

}

**if**(!validationResult.isSuccess()) {

**try** {

populateDropDowns(request);

} **catch** (SQLException | ClassNotFoundException throwables) {

throwables.printStackTrace();

}

request.setAttribute("errorString", validationResult.getError());

request.setAttribute("employee", employee);

RequestDispatcher dispatcher = **this**.getServletContext().getRequestDispatcher("/employee-form.jsp");

dispatcher.forward(request, response);

**return**;

}

**try** {

Employee entity = EmployeeMapper.MapToEntity(employee);

employeeRepository.InsertEmployee(entity);

Date currentDate = DateHelper.getUTCdatetimeAsDate();

java.sql.Date sqlDate = **new** java.sql.Date(currentDate.getTime());

positionHistoryRepository.InsertPositionHistory(**new** PositionHistory(sqlDate, **null**, employee.getId(), employee.getPositionId(), employee.getDepartmentId()));

} **catch** (SQLException | ClassNotFoundException throwables) {

throwables.printStackTrace();

}

response.sendRedirect("employee");

}

**protected** **void** **doGet**(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

**try** {

populateDropDowns(request);

} **catch** (SQLException | ClassNotFoundException throwables) {

throwables.printStackTrace();

}

RequestDispatcher dispatcher = request.getRequestDispatcher("employee-form.jsp");

dispatcher.forward(request, response);

}

**private** **void** **populateDropDowns**(HttpServletRequest request) **throws** SQLException, ClassNotFoundException {

List<LookupViewModel> statuses = **new** ArrayList<>();

statuses.add(**new** LookupViewModel(Constants.ResourceStrings.ActiveStatus, Integer.toString(EmployeeStatuses.Active)));

statuses.add(**new** LookupViewModel(Constants.ResourceStrings.BlockedStatus, Integer.toString(EmployeeStatuses.Blocked)));

request.setAttribute("statuses", statuses);

List<LookupViewModel> departments = **new** ArrayList<>();

List<Department> departmentEntities = departmentRepository.GetDepartments();

**for**(Department department : departmentEntities) {

departments.add(**new** LookupViewModel(department.getName(), Integer.toString(department.getId())));

}

request.setAttribute("departments", departments);

List<LookupViewModel> positions = **new** ArrayList<>();

List<Position> positionEntities = positionRepository.GetPositions();

**for**(Position position : positionEntities) {

positions.add(**new** LookupViewModel(position.getTitle(), Integer.toString(position.getId())));

}

request.setAttribute("positions", positions);

}

**private** EmployeeViewModel **parseForm**(HttpServletRequest request) {

EmployeeViewModel employee = **new** EmployeeViewModel();

employee.setFirstName(request.getParameter("firstName"));

employee.setLastName(request.getParameter("lastName"));

employee.setPatronymic(request.getParameter("patronymic"));

employee.setPhoneNumber(request.getParameter("phoneNumber"));

Date currentDate = DateHelper.getUTCdatetimeAsDate();

employee.setHireDate(**new** java.sql.Date(currentDate.getTime()));

**if**(request.getParameter("salary") != **null** && !request.getParameter("salary").equals(""))

employee.setSalary(Float.parseFloat(request.getParameter("salary")));

employee.setEmail(request.getParameter("email"));

employee.setUserName(request.getParameter("userName"));

employee.setPassword(request.getParameter("password"));

employee.setAdmin(**false**);

employee.setStatus(EmployeeStatuses.Active);

employee.setPositionId(Integer.parseInt(request.getParameter("positionId")));

employee.setDepartmentId(Integer.parseInt(request.getParameter("departmentId")));

**return** employee;

}

}