

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. каф. ЭВМ
_____ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему
ЛИЧНЫЙ КАБИНЕТ СОТРУДНИКА БГУИР

БГУИР ДП 1–40 02 01 01 307 ПЗ

Студент	А.О. Игнатович
Руководитель	И.Л. Селезнев
Консультанты:	
от кафедры ЭВМ	И.Л. Селезнев
по экономической части	Т.А Рыковская
Нормоконтролер	Е.Е. Клинецвич
Рецензент	

МИНСК 2023

РЕФЕРАТ

Дипломный проект представлен следующим образом. Электронные носители: 1 диск. Чертёжный материал: 6 листов формата А1. Пояснительная записка: 109 страницы, 18 рисунков, 3 таблиц, 10 литературных источников, 3 приложения.

Ключевые слова: веб-приложение, клиент, сервер, база данных, рабочий кабинет, списки, оповещения, заявления, авторизация, коммуникации.

Предметная область находится в кругу задач управления человеческими ресурсами, документооборотом и соцсетями.

Объектом разработки является веб-приложение.

Целью данного дипломного проекта является проектирование и реализация личного кабинета сотрудника БГУИР.

При разработке был использован язык Java, а именно некоторые модули фреймворка Spring, среда разработки IntelliJ IDEA. В качестве системы управления базами данных была выбрана PostgreSQL.

В результате разработки реализовано веб-приложение, предоставляющее возможность сотрудникам авторизоваться, видеть основную информацию о себе, видеть информацию о сотрудниках в целом в виде справочника, обмениваться оповещениями о предстоящих событиях, назначать задания, составлять заявления.

Данная система может быть использована для оптимизации рабочих процессов среди сотрудников университета, экономии их времени на поиске необходимой информации о сотрудниках и упрощению коммуникации между ними.

Разработанное приложение можно считать экономически эффективным. Оно позволяет рационально использовать запасы человеческих ресурсов, что способствует экономии их времени и сил.

Дипломный проект, с представленным функционалом, реализован в полном объёме. Также он имеет возможность для дальнейшего улучшения и расширения, посредством добавления большего функционала и интеграции с остальными модулями, используемыми в университете.

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 40 02 01-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

_____ Б.В.Никульшин

« ____ » _____ 2023 г.

ЗАДАНИЕ

по дипломному проекту студента

Игнатович Анны Олеговны

1 Тема проекта: «Личный кабинет сотрудника БГУИР» – утверждена приказом по университету от 3 апреля 2023 г. № 814-с.

2 Срок сдачи студентом законченного проекта: 1 июня 2023 г.

3 Исходные данные к проекту:

3.1 Протокол взаимодействия: HTTP, HTTPS.

3.2 Формат обмена данными: JSON.

3.3 Операционная система: Windows 10 Pro.

3.4 Среда разработки: IntelliJ IDEA.

3.5 Языки программирования: Java, React.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Техничко-экономическое обоснование разработки. Заключение. Список использованных источников. Приложения.

5 Перечень графического материала (с точным указанием обязательных чертежей):

5.1 Вводный плакат. Плакат.

- 5.2 Личный кабинет сотрудника БГУИР. Схема структурная.
 5.3 Личный кабинет сотрудника БГУИР. Диаграмма классов.
 5.4 Личный кабинет сотрудника БГУИР. Диаграмма последовательности.
 5.5 Личный кабинет сотрудника БГУИР. Схема программы.
 5.6 Заключительный плакат. Плакат.

6 Содержание задания по экономической части:

- 6.1 Расчет затрат на разработку личного кабинета сотрудника БГУИР.
 6.2 Оценка экономического эффекта проекта личного кабинета сотрудника БГУИР.

ЗАДАНИЕ ВЫДАЛ

Т.А Рыковская

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы. Сравнение аналогов. Уточнение задания на ДП	10	23.03 – 30.03	
Структурное проектирование	15	30.03 – 08.04	
Функциональное проектирование	25	08.04 – 24.04	
Разработка программных модулей	20	24.04 – 08.05	
Программа и методика испытаний	10	08.05 – 15.05	
Расчет экономической эффективности	5	15.04 – 20.05	
Оформление пояснительной записки	15	20.05 – 30.05	

Дата выдачи задания: 23.03.2023

Руководитель

И.Л. Селезнев

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1 ОБЗОР ЛИТЕРАТУРЫ	7
1.1 Обзор аналогов	7
1.2 Обзор технологий.....	9
1.3 Постановка задачи	16
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	17
2.1 Блок базы данных	17
2.2 Блок взаимодействия с базой данных	18
2.3 Блок пользовательского интерфейса	18
2.4 Блок взаимодействия пользовательского интерфейса с контроллерами запросов.....	19
2.5 Блок классов-контроллеров для обработки запросов	20
2.6 Блок бизнес-логики	21
2.7 Блок авторизации пользователей	22
2.8 Блок USER	23
2.9 Блок ADMIN	23
2.10 Блок BLOCK	23
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	24
3.1 Описание структуры приложения	24
3.2 Описание модели данных.....	27
3.3 Описание структуры и взаимодействия между классами.....	35
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	49
4.1 Алгоритм авторизации и регистрации	50
4.2 Работа Spring Security	54
4.3 Алгоритм запросов	57
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ	61
5.1 Модульное тестирование	64
5.2 Сквозное тестирование	64
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	65
6.1 Требования к аппаратному обеспечению	65
6.2 Руководство по развертыванию приложения	65
6.3 Руководство по использованию программного обеспечения	67
7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ЛИЧНОГО КАБИНЕТА СОТРУДНИКА БГУИР	74
7.1 Характеристика разработанного по индивидуальному заказу веб-приложения	74
7.2 Расчет затрат на разработку и цены программного средства, созданного по индивидуальному заказу	74
7.3 Расчет результата от разработки и реализации программного модуля личного кабинета сотрудника БГУИР	78
7.4 Расчет показателей экономической эффективности разработки программного модуля личного кабинета сотрудника БГУИР	79

ЗАКЛЮЧЕНИЕ.....	81
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	82
ПРИЛОЖЕНИЕ А	83
ПРИЛОЖЕНИЕ Б	107
ПРИЛОЖЕНИЕ В	109

ВВЕДЕНИЕ

В настоящее время использование информационных технологий в управлении человеческими ресурсами является одним из ключевых трендов в развитии современных организаций. Важным инструментом в этой сфере является личный кабинет сотрудника, который предоставляет возможность эффективного управления персоналом, повышения производительности и облегчения взаимодействия пользователей.

Личные кабинеты для сотрудников широко используются в сфере банковской и финансовой деятельности, где они позволяют управлять финансовыми данными, зарплатой, налогами и другими аспектами своей работы.

Цель данного дипломного проекта заключается в разработке личного кабинета сотрудника для Белорусского государственного университета информатики и радиоэлектроники (БГУИР).

Данная разработка предусматривает создание удобной и функциональной системы, которая позволит сотрудникам университета получать доступ к различного рода информации, связанной с рабочим процессом, помогает оптимизировать коммуникации между работниками и отделами, что способствует повышению эффективности и качеству взаимодействия. Разработка обеспечивает пользователей разными правами доступа, помогая защитить данные от незапланированных изменений путем разделения привилегий.

В дипломном проекте рассмотрены основные требования к личному кабинету сотрудника и проведен анализ существующих решений в данной области. Проанализирована внутренняя структура взаимодействия сотрудников университета, отделов и принципы их работы для наилучшего понимания и разработки необходимого функционала кабинета сотрудника, а также формирование фундамента для последующего расширения в многофункциональную систему.

В ходе разработки дипломного проекта, был определен ряд задач, которые рассматриваются в пояснительной записке:

1. Выбор и обоснование средств разработки системы.
2. Проектирование базы данных.
3. Разработка бизнес-логики серверной части.
4. Разработка логики взаимодействия клиентской части с серверной.
5. Разработка пользовательского интерфейса.
6. Тестирование системы.
7. Написание руководства пользователя.
8. Обоснование экономических затрат на разработку проекта.

Результатом выполнения вышеперечисленных задач является веб-приложение «Личный кабинет сотрудника БГУИР».

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

В данном разделе рассматриваются исследования предметных областей, которые затрагиваются в разрабатываемом проекте. Под предметными областями подразумевается используемые методы для создания, а также инструменты и подходы к проектированию.

В качестве вводной части дается рассмотреть используемые методы и технологии, которые применяются в дипломном проекте. В данном подразделе рассматривается и проводится анализ существующих аналогов, для разработки плана действий и облегчения поиска необходимых материалов. Необходимость состоит в предотвращении ошибок в проектировании и поиске наиболее оптимальных решений.

1.1.1 Sage HR

Sage HR [1] (см. рисунок 1.1) – это компания по разработке программного обеспечения (ПО) для управления персоналом. Приложение предназначено как для малых, так и для средних компаний. Управление происходит через веб-сайт и мобильное приложение, каждый пользователь имеет доступ к своему расписанию.

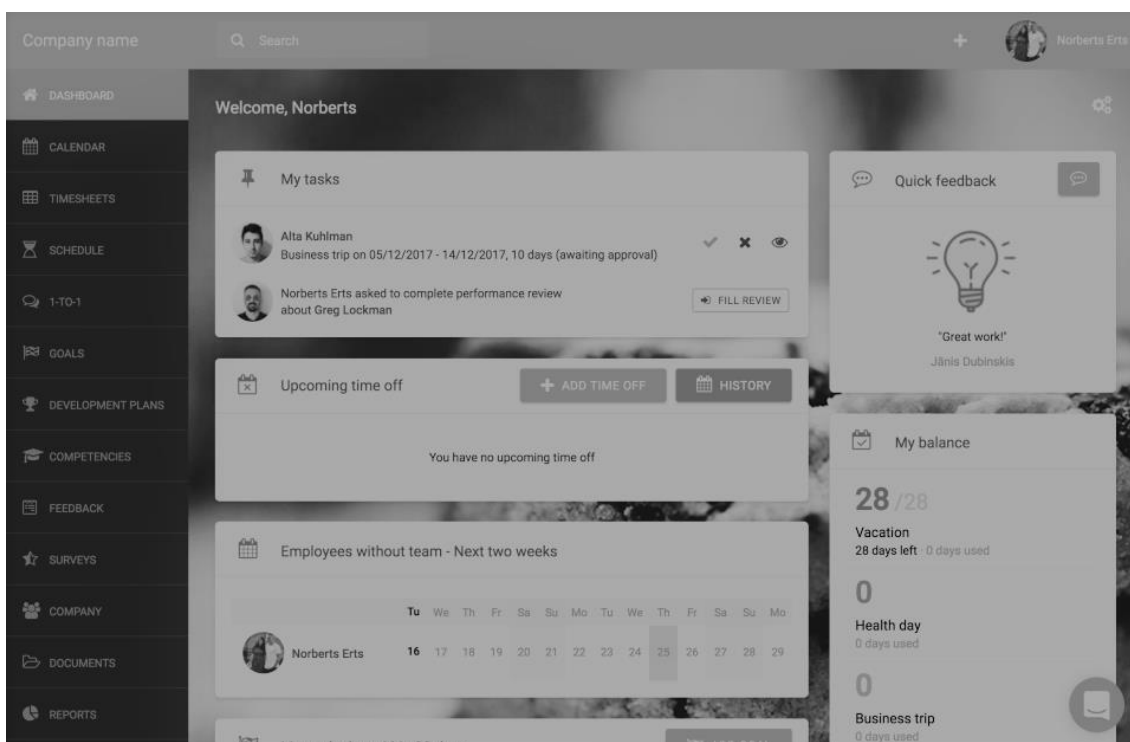


Рисунок 1.1 – Главная страница Sage HR [1]

На главной странице приложения представлены основные функциональные элементы, необходимые для работы пользователя. Дизайн

интерфейса привлекателен и выдержан в ярких тонах, однако, избыточное использование насыщенных цветов может вызвать утомление глаз при продолжительном использовании приложения.

Отсутствие русской локализации хоть и не является минусом, но все же это небольшой недостаток в силу того, что в университете много людей, которым предпочтительнее использовать русский.

При подключении данного приложения есть возможность выбирать только необходимые функции. Это значит, что программное средство модульное и каждая из его функциональных частей работает независимо от другой. Это достоинство, которое стоит иметь ввиду, потому что не весь функционал необходим сразу и возможность подключать по надобности достаточно удобная и практичная.

В данном приложении сотрудник должен самостоятельно регистрироваться. С одной стороны это плюс, так как может сэкономить время отделу кадров, но с другой, если возникнут ошибки при заполнении, возникнут разногласия в данных. В случае данного дипломного проекта эта функция возлагается на отдел кадров и администраторов, которые создают учетную запись работнику при его устройстве.

1.1.2 WebHR

WebHR [2] (см. рисунок 1.2) – это онлайн-инструмент для управления персоналом, который охватывает все аспекты работы отдела кадров, от приема на работу до выхода на пенсию. WebHR упрощает HR-процессы за счет шаблонизации, автоматизации и интеграции HR-процессов, чтобы к ним можно было получить доступ и обработать их с единой панели управления.

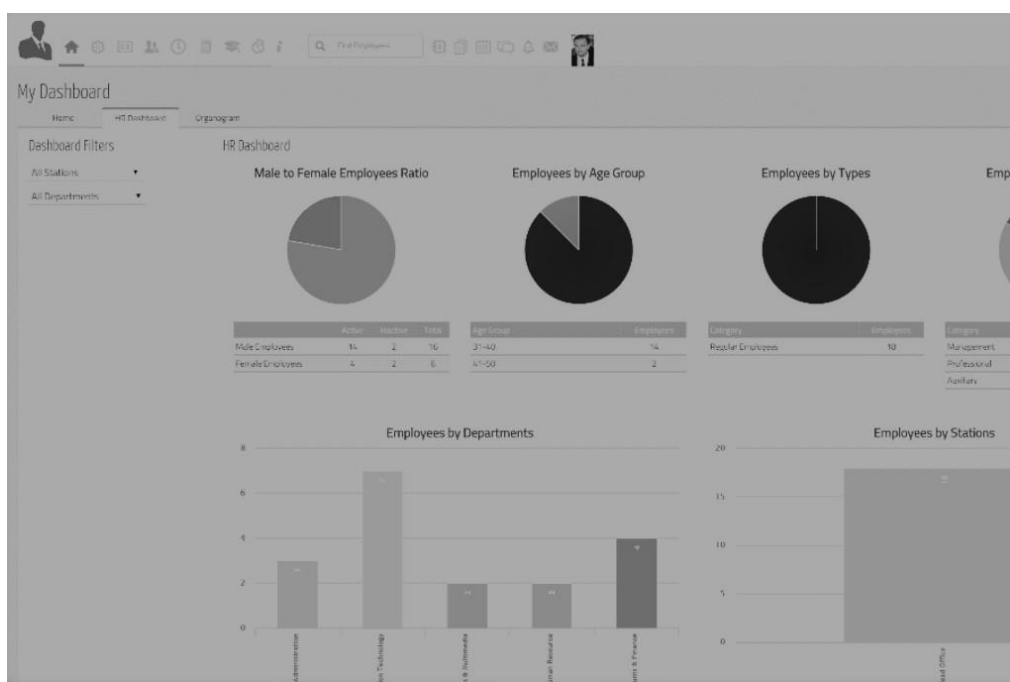


Рисунок 1.2 – Страница WebHR [2]

Одной из возможностей WebHR является интеграция с сайтом поиска работы Indeed.com, которая позволяет автоматически загружать объявления о вакансиях на сайт. Подобная функция удобная, но стоит учесть, что в нашем регионе используется другой сайт для поиска кадров. Стоит принять ее во внимание и учесть в дальнейшей разработке и развитии приложения.

За расчет заработной платы отвечает внешний подрядчик, однако все функции для расчета заработной платы также присутствуют в приложении. Тем не менее, следует отметить, что это приложение ориентировано преимущественно на работу с отделом кадров, поэтому это может стать недостатком в контексте необходимости обеспечения эффективного обмена информацией между разными подразделениями и, соответственно, самом использовании разными отделами. Одна из целей данного дипломного проекта заключается в обеспечении коммуникации между всеми отделами, так как разные отделы находятся в разных корпусах, поэтому ориентирование вокруг одного специализированного отдела не соответствует запросам данного веб-приложения.

Следует отметить, что наличие только англоязычной версии является недостатком приложения. Внедрение русской локализации в приложении может значительно улучшить его удобство и доступность для пользователей.

С точки зрения визуального оформления, это приложение выглядит более привлекательно, чем первый вариант, благодаря использованию простых и неярких цветов, которые не отвлекают внимание и позволяют ясно видеть все необходимые разделы и использования минималистичного дизайна.

Значительным же недостатком данного приложения является ограниченная поддержка. В БГУИР находится множество сотрудников и в случае возникновения сбоев в работе приложения, это может стать источником коллизий в рабочем процессе.

1.2 Обзор технологий

Дипломный проект предполагает разработку системы, которая является веб-приложением, это значит, что это ПО запускается в веб-браузере и оно имеет клиент-серверную архитектуру. Логика такого приложения разделена между сервером и клиентом, а данные, как правило, хранятся на сервере. Обмен информацией между сервером и клиентом происходит через сеть Интернет. Одним из основных преимуществ такого подхода является возможность запуска приложения на разных операционных системах, поскольку клиенты не зависят от конкретной ОС пользователя. В результате веб-приложения могут быть использованы на разных платформах, что делает их универсальными [3].

Система является небольшой и создается одним человеком, поэтому она имеет монолитную архитектуру. Это означает, что различные компоненты, а именно бизнес-логика, слой доступа к данным, он же подключение к базе

данных, интерфейс пользователя и так далее, находятся внутри одного процесса. Даная архитектура проста в развертывании, масштабировании и, соответственно, тестировании.

Паттерн проектирования, использованный в веб-приложении – Model, View, Controller (MVC). Приложение разделено на три условные части:

- модель, то есть данные, которые будут передаваться между представлениями и контроллерами;
- представления, которые будут визуализировать данные модели для пользовательского интерфейса;
- контроллеры, которые будут обрабатывать запросы и выбирать соответствующие им представления для визуализации [6].

Язык программирования для серверной части – Java, библиотека Spring. Этот фреймворк содержит много разных модулей. Для клиентской части – Java Script, библиотека React. База данных, для содержания необходимых данных – PostgreSQL.

1.2.1 Клиент-серверная архитектура

Клиент-серверная архитектура (см. рисунок 1.3) – это модель взаимодействия между компьютерами в сети, при которой один компьютер (сервер) предоставляет определенные ресурсы или услуги, а другие компьютеры (клиенты) запрашивают эти ресурсы или услуги через сеть.

В клиент-серверной архитектуре обычно выделяют два типа компонентов: клиентские и серверные.

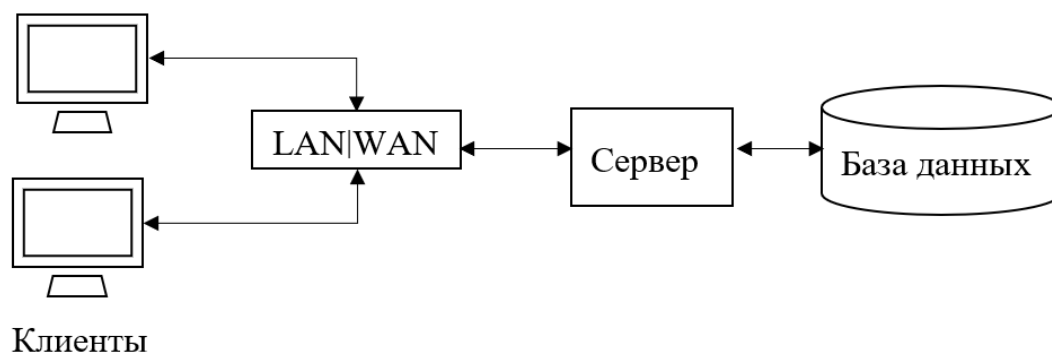


Рисунок 1.3 – Клиент-серверная архитектура

Клиенты – это устройства, на которых запускается пользовательский интерфейс, которые запрашивают ресурсы или функции у серверов. Конечный пользователь, который использует приложение, и обычно оно работает на его компьютере или мобильном устройстве.

Серверы – это высокопроизводительные вычислительные устройства, на которых размещаются ресурсы и функции, которые клиенты запрашивают. Серверы могут выполнять различные задачи, например, обрабатывать запросы

клиентов, хранить и обрабатывать данные, осуществлять авторизацию и аутентификацию пользователей и так далее.

Взаимодействие между клиентами и серверами происходит через сеть, обычно с использованием протоколов передачи данных, таких как HTTP, FTP, SMTP и тому подобные. Клиентские компоненты отправляют запросы на сервер, и серверные компоненты отвечают на эти запросы, предоставляя запрошенные данные или услуги.

Клиент-серверная архитектура является широко распространенной и используется во многих областях, таких как веб-приложения, базы данных, игровые системы и т.д. Архитектура позволяет создавать масштабируемые и гибкие системы, которые могут обрабатывать большие объемы запросов и предоставлять доступ к данным и услугам из любой точки сети.

В разрабатываемой системе присутствует еще один компонент – база данных, программа, в которой хранятся все данные приложения. Благодаря ей данная архитектура является трехуровневой, потому что она состоит из трех компонентов.

В клиент-серверной архитектуре сервер играет роль не только компьютера, на котором работает приложение или сайт, но также является хранилищем всех данных. Клиенты не имеют прямого доступа к базе данных, что защищает их личную информацию и обеспечивает приватность других пользователей, например, в социальных сетях.

Клиенты обращаются к серверу за информацией, и если сервер считает, что пользователь имеет права на получение этой информации, то он ее предоставляет. Подобный подход гарантирует защиту личных данных других пользователей [4].

Трехуровневая архитектура информационных систем может быть улучшена путем добавления дополнительных серверов и преобразована в многоуровневую архитектуру. Такая виртуальная архитектура позволяет значительно увеличить эффективность работы информационных систем и оптимизировать использование программно-аппаратных ресурсов [5].

1.2.2 MVC

Шаблон MVC (см. рисунок 1.4) [6] – это шаблон проектирования веб-приложений, который включает в себя три отдельных компонента: модель данных приложения, пользовательский интерфейс и логику взаимодействия пользователя с системой. Подобный подход позволяет минимизировать воздействие на другие компоненты при модификации одного из них. Кроме того, MVC включает несколько мелких шаблонов, что делает его более гибким и удобным для использования.

Основной целью использования шаблона проектирования MVC является разделение бизнес-логики и данных от визуализации. Такое разделение упрощает повторное использование кода и облегчает сопровождение. Например, если необходимо добавить новое представление

данных, такое как JSON, XML, PDF или XLSX, к уже существующему маршруту, это можно сделать без изменений в бизнес-логике маршрута. А изменения визуализации не затрагивают бизнес-логику, а изменения бизнес-логики не влияют на визуализацию.

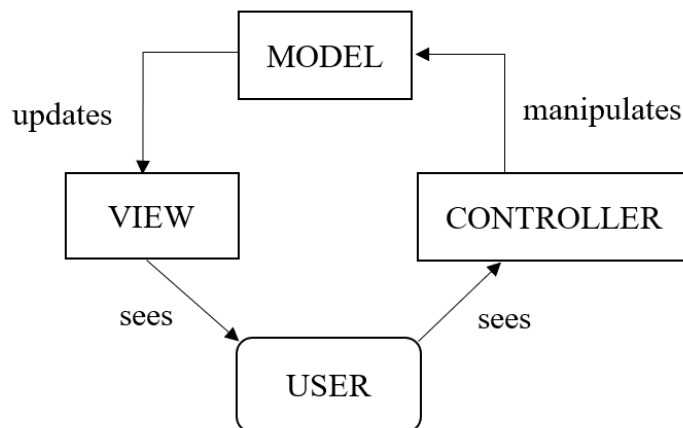


Рисунок 1.4 – Шаблон MVC [6]

Изменение каждого из модулей происходит независимо. Все модули имеют разные обязанности и ответственности и все поведение будет направлено на выполнение одной задачи. Подобный подход соответствует (single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion) SOLID -принципу единой ответственности [7].

1.2.3 База данных PostgreSQL

Для проекта была выбрана система управления базами данных PostgreSQL.

PostgreSQL – это популярная свободная объектно-реляционная система управления базами данных (СУБД). Система базируется на языке SQL и поддерживает многочисленные возможности [8].

Преимущества данной СУБД:

- поддержка баз данных неограниченного размера;
- мощные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
- наследование;
- легкая расширяемость.

СУБД обеспечивает высокую надежность и эффективность работы, включая поддержку транзакций (ACID) и встроенные механизмы репликации. Кроме того, система расширяема, что позволяет создавать собственные типы данных и индексы, а также расширять ее функциональность с помощью языков программирования.

Строительство системы на основе PostgreSQL обеспечивает более гладкую интеграцию с другими внутренними системами БГУИР, что является преимуществом для разработки данной системы.

1.2.4 Spring Framework Java

Spring Framework [9] – это один из самых популярных фреймворков для разработки приложений на языке Java, он универсальный, с открытым кодом. Предоставляет комплексный набор инструментов и библиотек для упрощения создания сложных приложений. В какой-то степени его можно назвать фреймворком фреймворков, потому что он состоит из отдельных модулей. В данном веб-приложении используются такие модули как Spring Data, Spring Security, Hibernate, Spring MVC, JPA.

Spring позволяет создавать RESRful веб-сервисы. Ниже эти модули рассмотрены подробнее.

REST (от англ. Representational State Transfer — «передача состояния представления») – это общие принципы организации взаимодействия приложения/сайта с сервером посредством протокола (HyperText Transfer Protocol) HTTP. По сути, взаимодействие с сервером происходит в четыре операции: 1) получение данных с сервера, 2) добавление на сервер новых данных, 3) изменение уже существующих на сервере данных и 4) удаление данных.

Основные принципы такой организации являются:

1. Ресурсы (Resources), каждый из которых должен обладать своим уникальным идентификатором URI (Uniform Resource Identifier), который клиент может использовать для доступа к этому ресурсу. Ресурсы могут представлять собой данные, которые могут быть получены или как-то изменены с помощью HTTP-методов.

2. Представления (Representation), когда каждый ресурс может иметь несколько представлений, которые определяют формат и содержание данных, возвращаемых сервером в ответ на запрос клиента. Представления могут быть в таких форматах как HTML, XML, JSON или любом другом формате, который может быть прочитан клиентом.

3. Методы HTTP (HTTP Methods), для работы с ресурсами, доступны четыре основных метода HTTP: GET для получения данных, POST для создания новых ресурсов, PUT для изменения уже существующих ресурсов и DELETE для удаления ресурсов.

4. Без состояния (Stateless), каждый запрос, отправленный клиентом на сервер, содержит все необходимые данные для его обработки. Сервер не запоминает информацию о предыдущих запросах клиента, что делает реализацию и масштабирование сервера более простым.

Spring Data – это набор инструментов и библиотек, предоставляющих единую модель программирования для доступа к данным. Набор использует

Spring-подход для упрощения работы с различными типами баз данных, включая реляционные, нереляционные и облачные базы данных.

Основной концепцией является репозиторий, который представляет собой набор интерфейсов для взаимодействия с JPA Entity. Например, интерфейс `CrudRepository<T, ID extends Serializable>`, который расширяет `Repository<T, ID>`, обеспечивает базовую функциональность для создания, чтения, обновления и удаления данных, иначе говоря, обеспечивают CRUD (create, read, update, delete).

Есть абстракции типа `PagingAndSortingRepository`, которая предоставляет методы для разбиения на страницы и сортировки записей. И еще одна `JpaRepository` предоставляет некоторые связанные с JPA методы, такие как очистка контекста постоянства и удаление записей в пакете.

Spring Security – это фреймворк безопасности, который предназначен для использования в приложениях, построенных на платформе Spring. Фреймворк предоставляет мощные функции аутентификации и авторизации, которые можно использовать для обеспечения безопасности веб-приложений, микросервисов, REST-сервисов и других приложений на основе Spring.

Основным компонентом Spring Security является фильтр безопасности, который работает на уровне HTTP запросов и ответов. Фильтр может быть настроен для выполнения различных задач, таких как проверка учетных данных пользователя, проверка разрешений доступа к ресурсам, фильтрация запросов и другое. Фильтр обеспечивает безопасность, путем применения цепочки фильтров к каждому запросу, что позволяет выполнять настройку защиты для каждого запроса.

Hibernate – это фреймворк для объектно-реляционного отображения в Java, который упрощает доступ к базам данных, позволяя разработчикам работать с объектами, а не с SQL-запросами. Hibernate позволяет сопоставлять объекты Java с таблицами в базе данных и автоматически генерировать соответствующий SQL-код, обеспечивает механизмы для управления отношениями между объектами, транзакциями, кэшированием данных и другими функциями, необходимыми при работе с базами данных.

Преимущества Hibernate включают улучшенную производительность, упрощенное программирование и снижение количества ошибок, связанных с неправильным использованием SQL-запросов. обеспечивает переносимость кода между различными СУБД и улучшенную безопасность при работе с базами данных.

Java Persistence API (JPA) – это спецификация Java EE для управления объектно-реляционным отображением в приложениях Java. Спецификация предоставляет стандартный способ описания сущностей, которые могут быть сохранены в базе данных, и управления их жизненным циклом в рамках приложения. JPA облегчает работу с базами данных и ORM-фреймворками, такими как Hibernate, EclipseLink, OpenJPA и другими.

JPA определяет аннотации, которые можно добавлять к классам и полям, чтобы указать отображение на реляционную базу данных. Эти аннотации включают аннотации, такие как `@Entity`, `@Table`, `@Column` и `@Id`, которые используются для описания сущностей и их свойств.

JPA также определяет набор методов для управления жизненным циклом объектов, таких как `persist()`, `merge()`, `remove()`, `refresh()` и `find()`. Поддерживает JPQL (Java Persistence Query Language), что позволяет создавать запросы к базе данных, используя объектную модель данных вместо SQL.

Обеспечивает абстракцию от конкретной базы данных, что позволяет легко переносить приложения между различными СУБД без изменения кода приложения. Это делает JPA мощным инструментом для разработки приложений, которые должны работать с различными базами данных и управлять большим объемом данных.

Spring MVC – это фреймворк для разработки веб-приложений на языке программирования Java, основанным на паттерне проектирования Model-View-Controller (MVC), описанном ранее в подразделе выше.

Фреймворк обеспечивает обработку ошибок, валидацию данных, аутентификацию и авторизацию, обработку AJAX-запросов и множество других функций, что делает его одним из наиболее популярных фреймворков для разработки веб-приложений на Java.

1.2.5 React

React [10] является JavaScript-библиотекой для создания пользовательских интерфейсов. Позволяет создавать компоненты, которые отвечают за отображение данных на странице. Компоненты могут быть многоразовыми и взаимодействовать между собой.

React использует DOM (Document Object Model или виртуальное дерево объектов), что позволяет обновлять только те элементы, которые действительно изменились, а не обновлять всю страницу целиком. Такой подход повышает производительность и скорость работы приложения.

DOM – это объектная модель документа, которая представляет собой иерархическую структуру документа в виде дерева объектов. Виртуальное DOM дерево состоит из узлов, которые могут быть элементами, атрибутами, текстом, комментариями и др. Узлы связаны друг с другом иерархически в родительские (`parentNode`) и дочерние (`childNodes`) отношения, а также соседние узлы (`previousSibling` и `nextSibling`). Позволяет программно создавать, изменять и удалять элементы и их атрибуты, а также реагировать на события, такие как щелчки мыши, изменения размеров и другие. DOM API может быть использован в JavaScript, чтобы создавать интерактивные и динамические веб-страницы.

Это не браузерная технология, это стандарт, определяемый World Wide Web Consortium (W3C). Браузеры предоставляют DOM API, который можно

использовать для манипулирования содержимым документа. DOM API позволяет получать доступ к элементам документа, изменять их содержимое, атрибуты и стили, добавлять и удалять элементы, а также реагировать на события.

Одним из преимуществ React является большое количество готовых компонентов и библиотек, которые можно использовать в своих проектах. React также хорошо подходит для создания больших приложений, которые могут быть разбиты на множество многократно используемых компонентов.

1.3 Постановка задачи

Исходя из анализа аналогов, можно сделать вывод, что для современных программ важна простота, незагруженный и интуитивно понятный интерфейс. Программа должна быть масштабируемая, платформенно независимая, чтобы ее можно было запускать на разных системах. Также следует отметить, что должен быть соответствующий нуждам пользователей функционал.

Применение указанных технологий и подходов поможет создать качественную систему для дипломного проекта, а также обеспечит выполнение требований к проекту.

Следует добавить, что данный дипломный проект создавался для учреждения, в котором уже существуют свои внутренние ресурсы и данная система должна не только соответствовать требованиям современного приложения, а также иметь возможность легко интегрироваться в имеющуюся среду. Соответственно, некоторыми из пунктов выполнения данного требования являлось использование СУБД PostgreSQL и инструментов Spring.

Основные принципы, лежащие в основе данного программного комплекса – это интуитивно понятный и легкий в использовании интерфейс, расширяемый функционал, а также охват нужд сотрудников и облегчение их коммуникаций.

Для дипломного проекта были определены следующие задачи:

- разработка системы таблиц в базе данных и их взаимодействие;
- разработка бизнес-логики для серверной части;
- разработка и реализация пользовательского интерфейса.

Данный программный комплекс будет реализован в виде веб-сайта, доступного через браузер, и предоставлять следующий набор функций:

- регистрация и авторизация пользователя;
- наличие трех уровней доступа – block, user и admin;
- просмотр всех сотрудников;
- администрирование аккаунтов;
- заполнение документов;
- создание событий и заданий, назначение их на других пользователей;
- система фильтрации списка пользователей для удобного поиска;
- смена логина.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Исходя из анализа теоретической части разрабатываемой системы был выделен ряд требований, которые необходимо выполнить для обеспечения стабильного и эффективного функционирования системы. Получив данный список, было принято решение разделить систему на функциональные блоки. Данный подход удобен тем, что внесение изменений в один блок не требует изменения системы в целом. Иными словами, изменяя один блок, остальные остаются нетронутыми и не нуждаются в правках, что значительно экономит время, а также позволит упростить и сделать сам процесс разработки удобнее.

Последующие блоки были выделены в данном дипломном проекте:

- блок базы данных;
- блок взаимодействия с базой данных;
- блок пользовательского интерфейса;
- блок взаимодействия пользовательского интерфейса с контроллерами;
- блок контроллеров запросов;
- блок бизнес-логики;
- блок авторизации пользователей;
- блок user;
- блок admin;
- блок block.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.307 С1.

Ниже рассматриваются вышеперечисленные блоки веб-приложения более подробно.

2.1 Блок базы данных

Стоит упомянуть, что СУБД PostgreSQL – это реляционная база данных, которая обладает множеством преимуществ перед другими СУБД. Эта база данных поддерживает Atomicity, Consistency, Isolation, Durability или же ACID-принципы, что обеспечивает надежность хранения данных. Также PostgreSQL обладает высокой производительностью благодаря использованию индексов и интеллектуальному планировщику запросов, а также расширяемости, которая позволяет пользователю определять новые функции и типы данных. Кроме того, PostgreSQL поддерживает язык SQL, а также JSON и имеет богатый набор типов данных. В целом, PostgreSQL является простой в использовании и гибкой СУБД, которая подходит для широкого спектра задач.

Реляционная база данных (РБД) – это база данных, основанная на реляционной модели данных. РБД состоит из таблиц (реляций), которые связаны между собой ключами. Каждая таблица в РБД представляет собой двумерную структуру, которая состоит из строк (кортежей) и столбцов

(атрибутов). Каждая строка представляет собой набор значений атрибутов, а каждый столбец определяет тип данных для данного атрибута.

Реляционные базы данных поддерживают операции CRUD (create, read, update, delete) для работы с данными, а также операции JOIN и GROUP BY для связывания данных из разных таблиц.

Данный блок представляет из себя модель данных или же реляционную базу данных, ее схему можно посмотреть на чертеже ГУИР.400201.307 PP2.

2.2 Блок взаимодействия с базой данных

Блок взаимодействия с базой данных – это важный компонент веб-приложения, который отвечает за связь между приложением и БД. Он позволяет сохранять, изменять и получать данные из БД.

Spring Data JPA является модулем Spring, который предоставляет удобный способ работы с базой данных через Java Persistence API (JPA). Этот модуль позволяет создавать репозитории для моделей данных приложения, которые автоматически будут преобразовываться в SQL-запросы, не требуя явного написания запросов. Spring Data JPA поддерживает различные реляционные базы данных, включая PostgreSQL, и предоставляет широкий спектр возможностей для работы с данными.

В случае данного дипломного проекта, данные передаются в формате JSON. Spring Data JPA предоставляет возможность преобразовывать данные в этот формат для сохранения в БД и обратно для получения данных из БД, что более просто и удобно позволяет взаимодействовать приложению и БД.

Spring Data JPA также позволяет использовать различные стратегии загрузки данных из БД. Жадная загрузка загружает все связанные объекты сразу при загрузке основного объекта, что может привести к увеличению нагрузки на БД и задержкам в работе приложения. Ленивая загрузка, напротив, загружает связанные объекты только при обращении к ним, что позволяет оптимизировать работу приложения и уменьшить нагрузку на БД. Такая загрузка также позволяет создавать динамические запросы на основе критериев. Это означает, что запросы формируются на основе условий, заданных во время выполнения приложения, что упрощает процесс написания запросов на языке SQL и делает код более читаемым и понятным.

2.3 Блок пользовательского интерфейса

Блок пользовательского интерфейса написан на React - библиотеке JavaScript для разработки интерфейсов. В интерфейсе используются компоненты, которые позволяют легко создавать и управлять элементами интерфейса.

Для создания стилей и макетов интерфейса используется CSS (Cascading Style Sheets) – это язык, используемый в веб-разработке для описания визуального представления документа, написанного на языке HTML

(Hypertext Markup Language). С помощью CSS можно определять стили элементов веб-страницы, такие как цвет, размер и расположение текста, фоновые изображения, отступы, рамки и т.д. CSS позволяет разделить содержимое документа и его представление, что позволяет создавать более гибкие и масштабируемые веб-страницы.

Используется React Bootstrap – это библиотека компонентов пользовательского интерфейса для React, которая использует компоненты Bootstrap, чтобы создавать дизайн и функциональность для приложений. React Bootstrap позволяет создавать адаптивный интерфейс для любых устройств, упрощает работу с CSS, и предоставляет удобный способ создания пользовательских компонентов.

Для обработки событий пользовательского взаимодействия и отправки запросов на сервер используется библиотека Axios. Библиотека позволяет делать запросы с различными параметрами, такими как заголовки, параметры запроса и тело запроса. Axios также умеет обрабатывать ошибки и возвращать данные в различных форматах, таких как JSON и XML. Это позволяет взаимодействовать с сервером более эффективно и удобно.

Библиотека React-router-dom используется для создания маршрутов веб-приложения и управления навигацией между страницами. Библиотека позволяет создавать ссылки, переходить на другие страницы, обрабатывать параметры запросов и отображать разные компоненты в зависимости от URL-адреса. Подобный подход позволяет создавать более сложные интерфейсы и улучшает навигацию в приложении.

Библиотека React-query используется для управления состоянием приложения и взаимодействия с сервером. Она позволяет делать запросы на сервер, кэшировать данные, обрабатывать ошибки и обновлять компоненты в зависимости от изменения данных. Это значительно упрощает разработку приложений и улучшает их производительность.

2.4 Блок взаимодействия пользовательского интерфейса с классами-контроллерами запросов

В приложении клиентская часть взаимодействует с серверной частью через REST API, который предоставляет классы-контроллеры приложения, то есть два класса, которые обрабатывают запросы клиентской части.

При отправке запроса с клиента на сервер, запрос сначала проходит через маршрутизатор в React-приложении, который определяет адрес и метод запроса. Затем запрос отправляется на соответствующий адрес на сервере.

На серверной стороне запрос обрабатывается контроллером, который содержит логику обработки запроса и взаимодействия с базой данных. Контроллер принимает запрос, выполняет нужные действия, получает или отправляет данные в базу данных через репозиторий, и возвращает ответ клиенту в виде JSON объекта.

Клиентская часть приложения принимает ответ от сервера и отображает его на странице при помощи React-компонентов. Если необходимо отправить данные на сервер, то клиентская часть будет формировать объект с данными и отправлять его на сервер через API. В свою очередь, серверная часть будет обрабатывать запрос и сохранять данные в базу данных.

2.5 Блок классов-контроллеров для обработки запросов

Блок контроллеров в приложении – это классы, которые отвечают за обработку HTTP запросов от клиента и передачу данных в сервисный слой приложения. Контроллеры реализованы на языке Java с использованием библиотеки Spring. Всего представлено два таких класса: `MainPageController` и `AuthenticationController`.

В классе `AuthenticationController` реализованы следующие запросы:

1. Запрос для аутентификации пользователей, принимает POST-запрос на пути `«/auth/authenticate»`. Контроллер проверяет корректность введенных пользователем данных и возвращает JWT токен в случае успешной аутентификации.

2. Запрос для входа и регистрации пользователей, POST-запрос на пути `«/auth/register»`. Контроллер проверяет корректность введенных пользователем данных и возвращает JWT токен в случае успешной регистрации.

В классе `MainPageController` реализованы следующие запросы:

1. Запрос для всей информации о сотруднике, которая будет отображаться на главной странице для авторизованного пользователя, POST-запрос на пути `«/mainUserInfo»`.

2. Запрос для всех заявлений, назначенных на авторизованного пользователя, которые будут отображаться на его странице, POST-запрос на пути `«/ls»`.

3. Запрос для подтверждения заявления, которое можно будет совершить с главной страницы, от авторизованного пользователя, GET-запрос на пути `«/ls/{id}»`.

4. Запрос для всех событий, назначенных на авторизованного пользователя и отображаемые его странице, POST-запрос на пути `/events»`.

5. Запрос для всех заданий, назначенных на авторизованного пользователя и отображаемые на его странице, POST-запрос на пути `«/tasks»`.

6. Запрос для просмотра общей информации о всех сотрудниках, POST-запрос на пути `«/employees»`.

7. Запрос для создания событий для пользователя, GET-запрос на пути `«/event»`.

8. Запрос для создания событий для пользователя, GET-запрос на пути `«/event»`.

9. Запрос для создания назначения событий для пользователя на других пользователей, GET-запрос на пути «/noticeevent/{idevent}».

10. Запрос для создания заданий для пользователя и назначения их на других пользователей, GET-запрос на пути «/task».

11. Запрос для авторизованного пользователя на смену своего пароля, GET-запрос на пути «/password».

12. Запрос для авторизованного пользователя на создание заявления, GET-запрос на пути «/lscreate».

13. Запрос для авторизованного пользователя с правами администратора на редактирование данных, GET-запрос на пути «/editemployee».

Формат всех передаваемых данных будет JSON.

2.6 Блок бизнес-логики

Блок бизнес-логики обрабатывает запросы, отправленные клиентом. Серверная часть использует библиотеки Spring, и представляет собой ряд возможностей:

1. Регистрацию и авторизацию пользователей: при регистрации нового пользователя производится проверка на уникальность логина и пароля. При авторизации пользователей проверяются истинность логина и пароля.

2. Администрирование аккаунта только пользователем с правами доступа ADMIN. Оно включает в себе манипуляцию данными сотрудников, а также возможность блокировать их аккаунты.

3. Создание заданий и событий: пользователи могут создавать задачи и события для других пользователей. При этом система отображает их на страницах определенных сотрудников, на которых они были назначены.

4. Управление документами: пользователи могут создавать и хранить документы в системе, а также имеют возможность отправлять их напрямую своим начальникам, а они в свою очередь могут их подтверждать или опровергать. Это поможет облегчить взаимодействие между сотрудниками.

5. Организация доступа: система будет предоставлять три уровня доступа – администратор, пользователь и заблокированный. Каждый уровень будет иметь свои права доступа к определенным функциям приложения.

6. Оповещения: пользователи будут получать оповещения о новых заданиях, просьбах, событиях и других событиях в системе в виде отображения на личной странице.

7. Фильтрация данных: система будет предоставлять функции фильтрации данных в списках сотрудников.

8. Работа с сотрудниками: приложение будет предоставлять возможность создания профиля для каждого сотрудника, хранения и управления персональной информацией, а также управления их задачами, событиями и заявлениями.

9. Возможность для каждого аккаунта самостоятельно сменить пароль.

10. При создании все пользователи по умолчанию будут создаваться как обычные пользователи. Возможность стать администратором может быть либо назначена с помощью другого администратора, либо напрямую через базу данных. Второй вариант не приветствуется из-за безопасности, но в чрезвычайных случаях имеет место быть.

В целом, блок бизнес-логики будет включать в себя все функции, необходимые для эффективной организации работы сотрудников университета, управлению их задачами и документами и организации их коммуникации.

2.7 Блок авторизации пользователей

Блок авторизации пользователей в веб-приложении – это критически важный блок, который обеспечивает безопасность доступа к системе. В данном проекте для реализации авторизации используется Spring Security, который предоставляет нам множество инструментов для работы с безопасностью в приложениях.

В веб-приложении используется механизм авторизации на основе токенов JWT. Для этого настраивается специальный класс-конфигурация, в котором определяются правила доступа и их конфигурации. Также настроен процесс аутентификации, при котором проверяется истинность логина и пароля.

Когда пользователь успешно проходит процесс аутентификации, то для него генерируется JWT токен, который он будет использовать для доступа к системе. Токен содержит информацию о пользователе и время его действия. Такой подход удобен тем, что так как можно настроить токены так, что по истечению, например, некоторого времени пользователю будет необходимо пройти авторизацию снова.

При каждом запросе на сервер проверяется наличие токена в заголовке запроса и его валидность. Для этого создается и настраивается фильтр, называемый `doFilterInternal`, который проверяет токен: если он действителен, то пользователю разрешается доступ к запрашиваемому ресурсу.

Фильтр `securityFilterChain`, как и предыдущий, работает при каждом запросе и отвечает за конфигурацию безопасности, а именно за настройку Spring Security. Фильтр определяет правила безопасности для приложения и обрабатывает каждый запрос, проверяя, имеет ли пользователь доступ к запрашиваемому ресурсу. Если у пользователя нет прав доступа, то он будет перенаправлен на страницу аутентификации или получит ошибку доступа.

Таким образом, блок авторизации пользователей в веб-приложении с использованием Spring Security и JWT токенов позволяет обеспечить безопасность доступа к системе и защищать данные пользователей от несанкционированного доступа.

2.8 Блок USER

Блок пользователя в данном приложении предназначен для управления доступом к функционалу и предоставления возможности работать с данными. Для каждого пользователя в системе создается уникальный профиль, который содержит личные данные, такие как имя, фамилия, электронная почта, пароль, а также права доступа к функционалу приложения.

Пользователь может зарегистрироваться в системе и авторизоваться в ней, чтобы получить доступ к своей личной странице и функциям, которые ему разрешены в соответствии с его уровнем доступа.

Важной частью блока пользователя является система безопасности, которая обеспечивает защиту данных и доступа к функционалу приложения. Эта система включает в себя проверку подлинности пользователей, аутентификацию и авторизацию, а также защиту данных от несанкционированного доступа. Кроме того, внутри данной системы применяется система токенов для передачи информации о пользователе.

2.9 Блок ADMIN

Блок администратора в данном приложении предназначен для управления доступом к функционалу и предоставления возможности работать с данными. Блок представляет собой пользователя с расширенными возможностями, а именно:

1. Управление учетными записями сотрудников: администратор сможет создавать, удалять и редактировать учетные записи сотрудников, а также изменять их уровень доступа и блокировать аккаунты, если по какой-то причине система дала сбой или необходимо заблокировать сотрудника из-за какой-то неординарной причины.

2. Мониторинг сотрудников: администратор сможет просматривать более углубленную информацию о сотрудниках.

2.10 Блок BLOCK

Блок представляет собой заблокированного пользователя и в данном приложении предназначен для ограничения доступа к функционалу и работе с данными.

Подобный блок необходим по причине политики безопасности, чтобы пользователи, которые по любой возможной причине, не имеют прав доступа к данным не смогли ими не только всячески манипулировать, а также не могли их получить и видеть в целом.

Блок заблокированных пользователей обеспечивает дополнительный уровень управления доступом, дополнительный уровень безопасности данных и снижает риск случайных или злонамеренных изменений информации.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Описание структуры приложения

Этот раздел посвящен описанию работы и состава разрабатываемого программного продукта. В дальнейшем представлены взаимосвязи между различными классами программного обеспечения в виде диаграммы классов ГУИР.400201.307 РР.1.

Для разработки серверной части программного продукта выбраны несколько технологий, которые используются в качестве инструментов и средств для создания функциональной и надежной системы. Эти технологии являются ключевыми элементами в разработке приложения и включают в себя набор инструментов для работы с базами данных, обеспечения безопасности и аутентификации пользователей, а также для реализации бизнес-логики приложения. Каждая из выбранных технологий имеет свои особенности и преимущества, что позволяет создавать более эффективную и гибкую систему в соответствии с требованиями проекта. Данные технологии в общих чертах рассматриваются ниже.

1. Spring Boot, которая позволяет создавать веб-сервер и настраивать взаимодействие между различными классами приложения, он автоматически добавляет в проект все необходимые зависимости и настраивает их для работы вместе.

2. Maven, используется для настройки процесса сборки, упаковки и запуска приложения, иначе говоря, для автоматизации сборки проектов. Данная технология использует файлы конфигурации POM (Project Object Model), в которых содержится информация о проекте, его зависимостях, конфигурациях, плагинах и других параметрах. С помощью данного инструмента и осуществляется подключение технологий Spring, например.

3. Spring Web, для создания веб-приложений. Этот модуль фреймворка Spring предоставляет инструменты для разработки на языке Java. Предоставляет ряд абстракций и компонентов, которые позволяют создавать масштабируемые, гибкие и безопасные веб-приложения.

4. Spring Data JPA, для работы с базой данных. Предоставляет реализацию JPA, которая упрощает доступ к базе данных и сокращает объем кода, необходимого для создания репозиторий и выполнения операций с базой данных. Spring Data JPA позволяет автоматически генерировать репозитории, которые позволяют выполнять CRUD (Create, Read, Update, Delete) операции с объектами, не нужно писать много кода вручную.

5. Spring Security, для обеспечения защиты и авторизации пользователей в системе. Предоставляет инструменты для обеспечения безопасности веб-приложений на основе Java.

В данном приложении серверная часть приложения реализована на Java. Исходя из гексагональной архитектуры, в приложении классы делятся по

сисполняемому функционалу. Вся структура пакетов и их описание приведена ниже (см. рисунок 3.1):

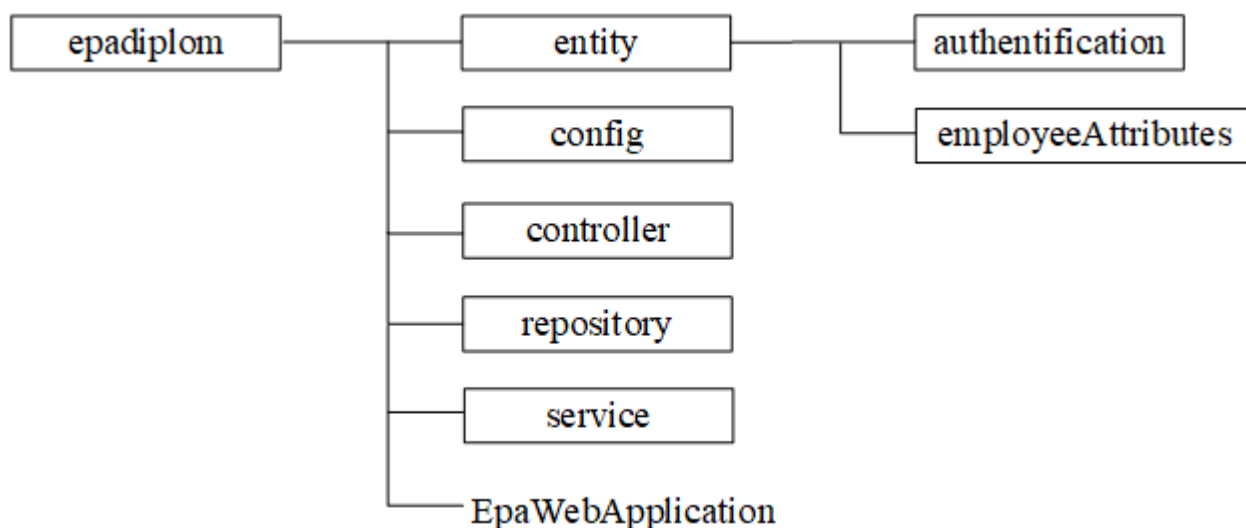


Рисунок 3.1 – Структура пакетов в веб-приложении

1. Пакет `config` – пакет, содержащий классы, которые отвечают за конфигурацию и настройку Spring Security, фреймворка, предназначенного для обеспечения безопасности приложений на платформе Spring. Данный пакет также отвечает за настройку JSON Web Token (JWT), механизма аутентификации и авторизации пользователей, использующего технологию передачи данных в формате JSON. Здесь происходит создание и настройка токенов, которые используются для идентификации пользователей и обеспечения безопасного доступа к ресурсам системы. Благодаря настройке JWT в данном пакете, система гарантирует безопасность передачи данных между клиентом и сервером и защищает от несанкционированного доступа. В него входят классы, перечисленные ниже:

- класс `ApplicationConfig`;
- класс `JwtAuthenticationFilter`;
- класс `SecurityConfig`.

2. Пакет `controller` – пакет, содержащий классы-контроллеры, которые являются частью паттерна проектирования Model-View-Controller (MVC). Контроллеры представляют собой классы, которые обрабатывают запросы от клиента и выполняют соответствующие действия в системе. Внутри каждого контроллера находятся методы-обработчики, которые реагируют на определенный тип запросов и возвращают клиенту соответствующий ответ. Данный пакет играет важную роль в обработке запросов и представляет собой основной механизм, с помощью которого клиент взаимодействует с системой. Содержит в себе классы:

- класс `AuthenticationController`;
- класс `MainPageConroller`.

3. Пакет `service` – пакет, содержащий классы, связанные с сервисом сущностей таблицы, а также касаются классов сервиса для обеспечения аутентификации, настройки токена. С помощью этих классов можно реализовать сложные запросы к таблицам, которые не могут быть выполнены с помощью `JpaRepository`. В сочетании с пакетами `repositories` и `entities`, они позволяют реализовать бизнес-логику приложения.

4. Пакет `repository` – пакет, содержащий интерфейсы, которые нужны для взаимодействия с базой данных с помощью репозитория, таких как `JpaRepository`. Это упрощает создание запросов к таблицам и в сочетании с пакетами `service` и `entities` позволяет реализовывать бизнес-логику приложения.

5. Пакет `entities` – пакет, содержащий сущности базы данных, которые используются вместе с пакетами `service` и `repository` для реализации бизнес-логики приложения. Он включает в себя 18 классов, из которых шесть являются представлениями, которые уже были упомянуты в модели данных и не нуждаются в дополнительном описании. Сущности, находящиеся в этом пакете, служат основой для работы с базой данных и представляют структуру данных, которые хранятся в ней. Вместе с классами из пакетов `service` и `repositories`, этот пакет обеспечивает полную реализацию бизнес-логики приложения. Также в пакете с сущностями содержится два подпакета – `authentication` и `employeeAttributes`.

5.1 Подпакет `authentication` – пакет, содержащий классы-сущности, которые отвечают за реализацию процесса аутентификации и авторизации пользователей в системе. В него входят классы, перечисленные ниже:

- класс `AuthenticationRequest`;
- класс `AuthenticationResponse`;
- класс `RegisterRequest`.

5.2 Подпакет `employeeAttributes` содержит список именованных констант – `role`, в котором находятся роли пользователей. Этот список используется для управления уровнем доступа пользователей при аутентификации и авторизации в системе. Различные роли дают пользователям различные уровни доступа.

Отдельно от всех этих моделей и пакетов находится класс `EraWebApplication` в общей папке со всем вышеперечисленным с названием `epadiplom`.

Подобная организация пакетов в приложении позволяет удобно добавлять новый функционал и вносить изменения в существующий код. При создании нового компонента можно просто создать новый пакет и добавить в него нужные классы, не затрагивая другие компоненты приложения. Такая организация обеспечивает изоляцию функционала и позволяет легко поддерживать код приложения. Кроме того, такая структура приложения упрощает работу команды разработчиков и ускоряет процесс разработки.

3.2 Описание модели данных

В данном разделе рассматривается база данных, которая работает с помощью СУБД PostgreSQL. Этот блок включает в себя данные, которые использует разрабатываемая система.

Для удобства модель данных можно условно разбить на несколько логических блоков, каждый из которых будет выполнять свою функцию в создаваемом веб-приложении (см. рисунок 3.2).

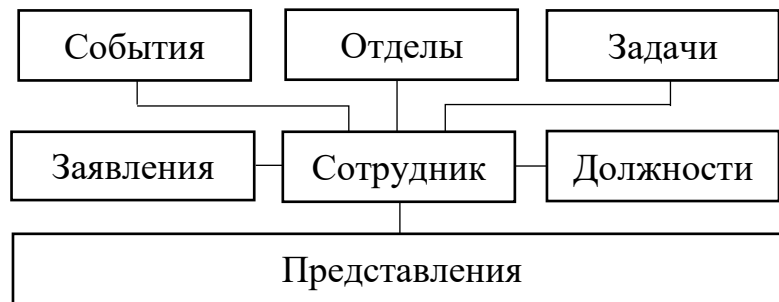


Рисунок 3.2 – Условное логическое разделение блоков БД

Данные сотрудника будут использоваться как для основной информации о сотруднике, так и хранить данные о пользователе, которые помогут получать информацию об определенном сотруднике.

Данные по событиям, задачам, отделам, заявлениям и должностям помогут получать информацию, исходя из их названий.

Все вышеперечисленные блоки представляют собой таблицы реляционной базы данных, которыми можно всячески манипулировать, но, в представленном логическом разделении, также присутствует блок представлений.

Представления являют собой виртуальные таблицы, которые будут помогать облегчать доступ к данным, не прибегая к созданию сложных запросов. Они нужны, если необходимо получить информацию из нескольких таблиц одновременно. Такие «таблицы» можно только просматривать без изменения данных. Чтобы их изменить придется обращаться к исходным таблицам.

3.2.1 Таблица Employee

Данная таблица предназначена для хранения основной информации о пользователе, которая можно указать в общем доступе, и которая будет отображаться в глобальном поиске сотрудников.

Поля таблицы:

- id – первичный ключ, bigint;
- first_name – имя сотрудника, varchar (128);
- middle_name – отчество сотрудника, varchar (128);

- last_name – фамилия сотрудника, varchar (128);
- work_number – рабочий номер сотрудника, numeric (16);
- location_street – зашифрованный пароль, который был выслан пользователю для восстановления аккаунта, varchar (128);
- cabinet_office – время и дата отправления пароля для восстановления аккаунта, varchar(8);
- id_dep – внешний ключ для связи с таблицей department, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем id используется специальный атрибут «null: false», это означает, что поле не может быть нулевым. Для остальных колонок это значение не выставлено, чтобы при создании аккаунта для сотрудника данные заполнялись отделом кадров и не было ошибок в системе и документах.

3.2.2 Таблица Login

Данная таблица предназначена для хранения информации о пользователе, которая связана с аккаунтом сотрудника, а именно содержит данные необходимые для авторизации и создания аккаунта.

Поля таблицы:

- id_login – первичный ключ, а также внешний ключ для таблицы employee, bigint;
- login_user – логин сотрудника, varchar (319);
- password_user – зашифрованный пароль, varchar (128);
- mail_user – мейл сотрудника, varchar (319);
- role – роль пользователя (касается аккаунта, а не работы), varchar (6).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем id_login, login_user, password_user, mail_user, role используется специальный атрибут «null:false», это означает, что поле не может быть нулевым. Все эти поля заполняются при создании профиля сотрудника.

3.2.3 Таблица Personal

Данная таблица предназначена для хранения личной информации о пользователе, которую можно будет увидеть только при наличии определенных прав, и которая требуется, в основном, отделу кадров.

Поля таблицы:

- id_personal – первичный ключ, а также внешний ключ для таблицы employee, bigint;
- birth_d – дата рождения сотрудника, date;
- entry_d – дата устройства на работу (в этот же день должен быть и создан аккаунт date);
- personal_number – личный номер сотрудника, numeric (16).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем `id_personal` используется специальный атрибут «`null: false`», это означает, что поле не может быть нулевым. Для остальных колонок это значение не выставлено, чтобы при создании аккаунта для сотрудника данные заполнялись отделом кадров и не было ошибок в системе и документах. Поле `entry_d` будет заполняться автоматически при создании аккаунта.

3.2.4 Таблица `Department`

Данная таблица нужна для содержания списка отделов университета, чтобы было удобнее распределять и сортировать сотрудников.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `name_dep` – название отдела университета, `varchar (128)`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полями `id` и `name_dep` используется специальный атрибут «`null: false`». Эти поля не могут быть нулевыми.

3.2.5 Таблица `Log_statement`

Данная таблица предназначена для информации о заявлениях. Они могут быть разных типов: от отпусков, до увольнения за свой счет и тому подобные.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `days_sum` – сумма дней, `integer`;
- `date_leave` – дата, когда работник уходит по заявлению (дата действия заявления), `date`;
- `date_of_ls` – дата, когда работник составляет заявление `date`;
- `id_approver` – номер сотрудника, который должен подтвердить заявление, `bigint`;
- `comment_ls` – комментарий сотрудника к заявлению, `varchar (300)`;
- `type_leave` – тип заявления (типы будут прописаны в логике), `smallint`;
- `status` – статус подтверждения, `numeric`;
- `id_employee` – внешний ключ для таблицы `employee` (того работника, что составляет заявление), `bigint`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля за исключением `comment_ls` используется специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении и сотрудник не может оставить их пустыми, за исключением комментария и самого скана документа. Поле `date_of_ls` будет заполняться автоматически при создании аккаунта.

3.2.6 Таблица Document

Данная таблица предназначена для хранения сканов оригинальных заявлений. Они не являются обязательными, поэтому таблица `log_statement` может существовать без привязки к данной таблице.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `id_ls` – внешний ключ для таблицы `log_statement`, `bigint`;
- `body_doc` – ссылка на скан оригинального заявления, `varchar (300)`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.7 Таблица Job_title

Данная таблица предназначена для хранения списка должностей сотрудников БГУИР.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `job_title_name` – название должности, `varchar (128)`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.8 Таблица Job_employee

Данная таблица является реализацией связи многие-ко-многим между таблицами `job_title` и `employee`.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `id_job_title` – внешний ключ для таблицы `job_title`, `bigint`;
- `id_employee` – внешний ключ для таблицы `employee`, `bigint`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.9 Таблица Task

Данная таблица предназначена для хранения информации о заданиях или же действиях, которые нужно сделать работнику.

Поля таблицы:

- `id` – первичный ключ, `bigint`;

- `date_task` – дата, когда работник составляет заявление, `date`;
- `name_of_task` – название задания или его суть, `varchar (128)`;
- `id_executor` – номер сотрудника, который будет исполнять задание, `bigint`;
- `comment_te` – комментарий сотрудника к заданию, по сути, описание, если таковое требуется, `varchar (300)`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля, кроме `comment_te`, используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении. Поле `comment_te` можно пропустить, потому что некоторые задания могут быть ясны без уточнений.

3.2.10 Таблица **Emp_task**

Данная таблица является реализацией связи многие-ко-многим между таблицами `task` и `employee`.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `id_job_title` – внешний ключ для таблицы `job_title`, `bigint`;
- `id_employee` – внешний ключ для таблицы `employee`, `bigint`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.11 Таблица **Event**

Данная таблица предназначена для хранения информации о событиях, созданных сотрудниками, а также назначения, для кого они предназначены.

Поля таблицы:

- `id` – первичный ключ, `bigint`;
- `date_of_event` – дата события, `date`;
- `type_of_event` – название события, `varchar (40)`;
- `comment_fe` – комментарий сотрудника к событию, по сути, описание, если таковое требуется, `varchar (300)`.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля, кроме `comment_fe`, используют специальный атрибут «`null: false`», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении. Поле `comment_fe` можно пропустить, потому что некоторые задания могут быть ясны без уточнений. Формат данных `timestamp without time zone` означает, что нет привязки к часовому поясу. Это сделано, чтобы не было разногласий во времени.

3.2.12 Таблица Notice_event

Данная таблица является реализацией связи многие-ко-многим между таблицами event и employee.

Поля данной таблицы:

- id – первичный ключ, bigint;
- id_event – внешний ключ для таблицы event, bigint;
- id_recipient – номер сотрудника, которому предназначается отправить событие, bigint;
- id_employee – внешний ключ для таблицы employee, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.13 Представление Employee_full_info_view

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых общей информации о сотруднике, которая представляет собой данные по аккаунту, персональные данные и общие сведения, которые будут в общем доступе.

Таблицы, которые объединены в данном представлении: personal, employee, department, job_employee, job_title, login.

Поля представления:

- job_employee.id;
- login.id_login;
- employee.first_name;
- employee.middle_name;
- employee.last_name;
- personal.birth_d;
- personal.entry_d;
- login.login_user;
- login.password_user;
- login.mail_user;
- login.role;
- employee.work_number;
- personal.personal_number;
- employee.location_street;
- employee.cabinet_office;
- department.name_dep;
- job_title.job_title_name.

3.2.14 Представление **Employees_view**

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых общей информации о сотруднике, которые могут быть в общем доступе и видимы для других сотрудников.

Таблицы, которые объединены в данном представлении: employee, department, job_employee, job_title.

Поля представления:

- job_employee.id;
- job_employee.id_employee;
- employee.first_name;
- employee.middle_name;
- employee.last_name;
- employee.work_number;
- employee.location_street;
- employee.cabinet_office;
- department.name_dep;
- job_title.job_title_name.

3.2.15 Представление **Ls_view**

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых заявлений и их заполнения.

Таблицы, которые объединены в данном представлении: employee, log_statement, document, login.

Поля представления:

- document.id;
- log_statement.type_leave;
- log_statement.date_leave;
- log_statement.date_of_ls;
- log_statement.days_sum;
- log_statement.id_approver;
- log_statement.status;
- log_statement.comment_ls;
- log_statement.id_employee;
- document.body_doc;
- login.role.

3.2.16 Представление **Events_view**

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых событий сотрудников университета.

Таблицы, которые объединены в данном представлении: `event`, `notice_event`. Поля представления:

- `notice_event.id`;
- `event.type_of_event`;
- `event.date_of_event`;
- `event.comment_fe`;
- `notice_event.id_recipient`;
- `notice_event.id_employee`.

3.2.17 Представление **Job_title_view**

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых должности сотрудников.

Таблицы, которые объединены в данном представлении: `job_title`, `job_employee`.

Поля представления:

- `job_employee.id`;
- `job_title.job_title_name`;
- `job_employee.id_employee`.

3.2.18 Представление **Tasks_view**

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых заданий, которые сотрудники могут назначать друг на друга.

Таблицы, которые объединены в данном представлении: `task`, `emp_task`.

Поля представления:

- `emp_task.id`;
- `emp_task.id_task`;
- `task.name_of_task`;
- `task.date_task`;
- `task.comment_te`;
- `task.id_executor`;
- `emp_task.id_employee`.

3.3 Описание структуры и взаимодействия между классами

При создании приложения использовался паттерн Model-View-Controller (MVC), который определяет его структуру, состоящую из трех основных компонентов: контроллера, сервиса и репозитория. Кроме того, следует отметить, что все созданные сервисы разработаны в соответствии с правилами и стандартами REST-архитектуры. Это касается серверной части приложения.

Контроллеры отвечают за обработку входящих HTTP-запросов и вызывают соответствующие методы сервисов, которые обрабатывают эти запросы и возвращают результаты. Сервисы представляют собой прослойку между контроллером и репозиторием и отвечают за бизнес-логику приложения, такую как проверка прав доступа, обработка данных и т.д. Репозитории служат для связи с базой данных и содержат методы для выполнения CRUD-операций (создание, чтение, обновление и удаление данных).

3.3.1 Класс `EpaWebApplication`

Точка входа для запуска веб-приложения на основе фреймворка Spring Boot. Аннотация `@SpringBootApplication` указывает, что это главный класс приложения и сообщает Spring, что нужно выполнить все необходимые конфигурации и инициализации для запуска веб-приложения.

Метод `main()` вызывает метод `run()` класса `SpringApplication`, который запускает приложение. В качестве аргументов метод `run()` принимает класс `EpaWebApplication` и аргументы командной строки `args`.

Таким образом, этот класс и его метод `main()` запускают Spring Boot приложение и начинают обработку входящих HTTP запросов.

3.3.2 Класс `ApplicationConfig`

Класс представляет собой конфигурационный класс Spring, который содержит конфигурацию для аутентификации пользователей в системе. В нем объявлены такие аннотации, как `@Component` указывает, что класс является компонентом Spring и должен быть автоматически сканирован и зарегистрирован в контексте Spring, классы, отмеченные этой аннотацией, могут быть внедрены (injected) в другие классы в качестве зависимостей, и аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В классе определены следующие поля, описанные ниже:

1. Класс `userDetailsService` возвращает сервис для поиска пользователей по имени пользователя, используя репозиторий `UserRepo`.

2. Класс `authenticationProvider` создает провайдера аутентификации `DaoAuthenticationProvider`, который использует `userDetailsService` для поиска пользователя в базе данных и `passwordEncoder` для проверки пароля пользователя.

3. Класс `authenticationManager` создает и возвращает менеджер аутентификации `AuthenticationManager`, используя конфигурацию аутентификации.

4. Класс `passwordEncoder` возвращает объект `BCryptPasswordEncoder`, который используется для хэширования пароля пользователя.

3.3.3 Класс `JwtAuthenticationFilter`

Данный класс представляет собой фильтр аутентификации, который будет вызван один раз для каждого запроса, прошедшего через контроллер в приложении. Фильтр проверяет наличие токена авторизации в заголовке запроса и, если он присутствует, использует сервис JWT для проверки его валидности и получения имени пользователя из токена. Затем фильтр проверяет, что пользователь существует в базе данных и, если это так, создает аутентификационный токен Spring Security и устанавливает его в контекст безопасности. Если токен авторизации не найден или недействителен, фильтр пропускает запрос и передает его дальше по цепочке фильтров.

В классе объявлены такие аннотации, как `@Component` указывает, что класс является компонентом Spring и должен быть автоматически сканирован и зарегистрирован в контексте Spring, классы, отмеченные этой аннотацией, могут быть внедрены (`injected`) в другие классы в качестве зависимостей, и аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В нем всего один метод, который выполняет все вышеперечисленное:

- `protected void doFilterInternal;`

Также в коде есть два поля, которые получает конструктор:

- `private final JwtService jwtService` – класс, который реализует логику работы с JWT токенами;
- `private final UserDetailsService userDetailsService` – сервис, который будет использоваться для загрузки информации о пользователе по логину.

3.3.4 Класс `SecurityConfig`

Этот класс содержит конфигурацию Spring Security для веб-приложения. Он использует аннотации Spring `@Configuration`, которая указывает, что этот класс является конфигурационным файлом Spring, `@EnableWebSecurity`, которая включает использование Spring Security в приложении и

`@RequiredArgsConstructor`, которая генерирует конструктор, использующий поля с аннотацией `@NonNull` как параметры конструктора.

Данный класс содержит методы:

- `securityFilterChain` – метод, который создает цепочку фильтров безопасности;
- `corsConfigurationSource` – метод, который создает и настраивает бин, который определяет конфигурацию Cross-Origin Resource Sharing (CORS);
- методы `sessionManagement()` и `sessionCreationPolicy()` устанавливают стратегию управления сессиями;
- метод `authenticationProvider()` конфигурирует провайдер аутентификации для приложения;
- метод `addFilterBefore()` добавляет фильтры перед указанным фильтром.

В данном классе еще есть два поля:

- `jwtAuthFilter` – это объект фильтра, который будет использоваться для проверки JWT-токенов и аутентификации пользователей;
- `authenticationProvider` – это объект, который будет использоваться для проверки учетных данных пользователей.

3.3.5 Класс `AuthenticationController`

Данный класс представляет контроллер для обработки запросов, связанных с аутентификацией и авторизацией пользователей.

Аннотация `@RestController` указывает на то, что класс предназначен для обработки HTTP-запросов, а возвращаемые им методы должны быть преобразованы в тело ответа HTTP.

Аннотация `@RequestMapping("/auth")` указывает на корневой путь, который будет использоваться для обработки запросов, обрабатываемых этим контроллером.

Аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В классе есть несколько методов, которые рассмотрены ниже:

1. Метод `register` – он обрабатывает POST-запросы на `/auth/register`. Он принимает в теле запроса объект `RegisterRequest`, содержащий данные, необходимые для регистрации нового пользователя, и передает их в сервис `AuthenticationService` для выполнения регистрации. Затем он возвращает объект `AuthenticationResponse`, содержащий информацию об успешности регистрации и авторизации нового пользователя.

2. Метод `authenticate` – он обрабатывает POST-запросы на `/auth/authenticate`. Он принимает в теле запроса объект

AuthenticationRequest, содержащий учетные данные пользователя (имя пользователя и пароль), и передает их в сервис AuthenticationService для выполнения аутентификации. Затем он возвращает объект AuthenticationResponse, содержащий JWT-токен, который пользователь может использовать для авторизации на защищенных ресурсах.

3.3.6 Класс MainPageController

Этот класс является контроллером Spring Boot и содержит обработчики HTTP-запросов. Он предназначен для работы с главной страницей приложения. Класс MainPageController использует несколько репозиторий для доступа к данным в базе данных, которые хранят информацию о сотрудниках, логах, событиях и других объектах. Каждый метод возвращает список объектов, который сериализуется в JSON и отправляется обратно клиенту в ответ на запрос. Также в этом классе есть методы, которые используют Spring Security для аутентификации пользователей и контроля доступа к данным.

Содержит такие аннотации, как @RestController, которая указывает на то, что класс предназначен для обработки HTTP-запросов, а возвращаемые им методы должны быть преобразованы в тело ответа HTTP, @EnableMethodSecurity, которая включает использование Spring Security в приложении, @EnableWebMvc, которая разрешает проекту использовать MVC, и @RequiredArgsConstructor от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией @NonNull или final, также автоматически создает методы hashCode(), equals(), и toString(), используя поля класса.

Содержит ряд методов, которые выполняют различные операции, вроде получения информации о сотрудниках, заявках, событиях и заявлениях, утверждение полученных заявок, а также создание событий, заявок и заявлений, смена пароля, получение информации о себе. Ниже будет список методов:

1. Метод `getEmployeeInfo()` – в этом методе контроллера происходит получение данных о авторизированном пользователе, которые содержатся в таблице `employee_full_view`. Метод `findAllByIdLogin()` выполняет выборку всех записей из этого представления.

2. Метод `getEmployees()` – этот метод возвращает список всех пользователей, зарегистрированных в системе. Запрос к базе данных выполняется с использованием метода `findAll()` из сервиса `employeesViewService`.

3. Метод `getLogRequests()` – этот метод получает список запросов на изменение данных (`log statements`), которые ожидают подтверждения со стороны пользователя. Выборка выполняется с использованием метода `findAllByIdApproverAndStatus()` из сервиса `logStatementViewService`.

Параметр `idApprover` указывает на идентификатор пользователя, которому требуется подтверждение изменений, а `status` задает статус запроса (1 - подтвержден, 2 - отклонен, 3 - требуется подтверждение).

4. Метод `getEvents()` – этот метод возвращает список всех событий, связанных с пользователем. Запрос выполняется с использованием метода `findAllByIdRecipient()` из сервиса `eventsViewService`. Параметр `idRecipient` указывает на идентификатор пользователя, для которого запрашиваются события.

5. Метод `setLsApprove()` – этот метод позволяет подтвердить или отклонить заявление сотрудника, обрабатывает POST-запрос в котором содержится объект `LogStatementRequest`, который содержит новый статус.

6. Метод `getTasks()` – этот метод обрабатывает GET-запрос на получение списка задач. Использует `id` для вызова метода `findAllByIdExecutor` у `tasksViewService`.

7. Метод `createEvent()` – этот метод создает новое событие, используя объект `EventRequest`, который содержит данные, необходимые для создания события.

8. Метод `createNoticeEvent()` – этот метод создает связь между событием и получателем, используя объект `NoticeEventRequest`, который содержит данные, необходимые для создания подобной таблицы.

9. Метод `createTask()` – этот метод создает новое задание, используя объект `TaskRequest`, который содержит данные, необходимые для создания задания.

10. Метод `changePassword()` – этот метод позволяет пользователю сменить его пароль, используя метод `setPassword()` и кодирует новый пароль с помощью `passwordEncoder`. Он использует объект типа `LoginRequest` в теле запроса, который содержит новый пароль.

10. Метод `changeLS()` – этот метод создает новую заявку, используя объект типа `LogStatementCreateRequest`, который содержит данные о заявлении и, если таковой имеется, то и о документе.

11. Метод `edit()` – этот метод позволяет изменять данные сотрудника, доступен только пользователям с соответствующим уровнем доступа.

Все методы принимают и\или отдают данные в формате JSON.

3.3.7 Классы сущностей

Далее идут классы сущностей, описывающих таблицы в базу данных. Они схожи по структуре и содержанию.

Аннотации `@Entity` и `@Table` указывают, что класс является сущностью JPA и что он будет сопоставлен с соответствующей таблицей в базе данных.

Аннотации `Lombok` `@Builder`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Getter` и `@Setter` добавляют готовые методы для

создания объектов, конструкторов без аргументов и с аргументами, геттеров и сеттеров для полей класса.

Аннотация `@JsonIgnoreProperties(ignoreUnknown = true)` указывает на то, что при десериализации JSON-объекта, все неизвестные свойства должны быть проигнорированы, а не вызывать ошибку десериализации. Это означает, что если в JSON-объекте есть дополнительные свойства, которые отсутствуют в Java-объекте, то они будут проигнорированы при десериализации и не будут помещены в Java-объект.

Аннотация `@JsonSerialize` указывает на то, что при сериализации Java-объекта в JSON-объект, должен использоваться определенный способ сериализации по умолчанию.

Аннотация `@Transactional` применяется тогда, когда метод, помеченный аннотацией, вызывается, Spring открывает транзакцию, выполняет метод, и, если метод завершается успешно, фиксирует транзакцию. Если же метод завершается неудачно (например, возникает исключение), транзакция откатывается, и все изменения в базе данных, сделанные в рамках этой транзакции, отменяются.

Также в классах описываются связи между таблицами с помощью аннотаций `@OneToMany`, `@ManyToOne` и `@OneToOne`. Ниже перечислены классы-сущности:

- класс `Department`;
- класс `Document`;
- класс `Employee`;
- класс `EmployeeFullView`;
- класс `EmployeesView`;
- класс `EmployeeTask`;
- класс `Event`;
- класс `EventsView`;
- класс `JobEmployee`;
- класс `JobTitle`;
- класс `LogStatement`;
- класс `LogStatementsView`;
- класс `TasksView`;
- класс `JobTitleView`;
- класс `NoticeEvent`;
- класс `Personal`;
- класс `Task`;
- класс `User`.

Подробнее рассмотрим сущность `User`. Она представляет собой таблицу `Login` из базы данных. Данная сущность нужна для реализации авторизации пользователя в системе, потому что позволяет манипулировать данными сотрудника и в принципе позволяет осуществлять связь конкретного пользователя с базой данных.

Класс `User` также реализует интерфейс `UserDetails` из `Spring Security`, который содержит методы для получения информации о пользователе, используемой при аутентификации и авторизации пользователей в приложении.

В частности, методы `getAuthorities()`, `getPassword()`, `getUsername()` используются для проверки прав доступа пользователей в системе, а методы `isAccountNonExpired()`, `isAccountNonLocked()`, `isCredentialsNonExpired()`, `isEnabled()` позволяют проверять статусы учетной записи пользователя.

Также, этот класс определяет соответствующие поля и методы для работы с базой данных, используя аннотации `JPA`.

3.3.8 Интерфейсы репозитория

Теперь рассмотрим интерфейсы, являющиеся репозиториями для сущностей, описанных ранее. Они наследуются от `JpaRepository<Repo, Long>`, что позволяет ему использовать стандартные методы доступа к данным (CRUD), такие как сохранение, обновление, удаление, поиск и так далее.

Ниже перечислены все интерфейсы для классов-сущностей: Данные интерфейсы предоставляют методы для выполнения операций чтения/записи данных. Классы сервиса используют эти методы репозитория для выполнения бизнес-логики и обработки запросов от контроллера, а затем возвращают результат контроллеру для отображения в пользовательском интерфейсе.

Таким образом, классы сервисов и репозитория взаимодействуют друг с другом, обеспечивая разделение ответственностей между слоями приложения и упрощая его тестирование и сопровождение.

Аннотация `@Repository` указывает, что интерфейс является репозиторием, который может управлять базой данных и извлекать из нее данные.

- интерфейс `DepartmentRepo`;
- интерфейс `DocumentRepo`;
- интерфейс `EmployeeFullViewRepo`;
- интерфейс `EmployeeRepo`;
- интерфейс `EmployeesViewRepo`;
- интерфейс `EmployeeTaskRepo`;
- интерфейс `EventRepo`;
- интерфейс `EventsViewRepo`;
- интерфейс `JobEmployeeRepo`;
- интерфейс `JobTitleRepo`;
- интерфейс `LogStatementRepo`;
- интерфейс `LogStatementsViewRepo`;
- интерфейс `TasksViewRepo`;

- интерфейс JobTitleViewRepo;
- интерфейс NoticeEventRepo;
- интерфейс PersonalRepo;
- интерфейс TaskRepo;
- интерфейс UserRepo.

Благодаря подобным репозиториям создаются запросы к сущностям прямо из имени метода, используя специальные префиксы типа `find...By`, `read...By`, `query...By`, `count...By`, и `get...By` и тому подобные. Кроме них можно использовать дополнительные выражения, которые усложняют запросы и уточняют выборку необходимых данных.

3.3.9 Классы сервиса

Теперь рассмотрим классы сервиса. Данные классы представляют сервисные компоненты, которые предназначены для работы с сущностями в системе. Они используют интерфейсы репозитория для доступа к базе данных и предоставляют методы для выполнения операций с сущностями, такие как сохранение, обновление, удаление и получение данных.

Далее будут представлены списки всех классов, используемых для сервиса, которые связаны с сущностями:

- класс `DepartmentService`;
- класс `DocumentService`;
- класс `EmployeeService`;
- класс `EmployeeFullViewService`;
- класс `EmployeesService`;
- класс `EmployeeTaskService`;
- класс `EventService`;
- класс `EventsViewService`;
- класс `JobEmployeeService`;
- класс `JobTitleService`;
- класс `JobTitleViewService`;
- класс `LogStatementService`;
- класс `LogStatementsViewService`;
- класс `NoticeEventService`;
- класс `PersonalService`;
- класс `TaskService`;
- класс `TasksViewService`;
- класс `UserService`.

Кроме вышеперечисленных классов имеется два специальных класса сервиса `JwtService` и `AuthenticationService`. Классы рассмотрим отдельными пунктами.

3.3.10 Класс `JwtService`

Этот класс предоставляет функционал для генерации и проверки JSON Web Tokens (JWT), которые используются для аутентификации пользователей в приложениях.

Данный класс содержит следующие методы:

- `extractUsername(jwtToken)` – извлекает имя пользователя из JWT;
- `extractClaim(jwtToken, claimsResolver)` – извлекает любое утверждение из JWT, используя переданный функциональный интерфейс `claimsResolver`;
- `generateToken(userDetails)` – генерирует JWT для пользователя `userDetails`;
- `isTokenValid(jwtToken, userDetails)` – проверяет, действителен ли JWT для пользователя `userDetails`;
- `isTokenExpired(jwtToken)` – проверяет, истекло ли время жизни JWT;
- `extractExpiration(jwtToken)` – извлекает дату истечения срока действия JWT;
- `extractAllClaims(jwtToken)` – извлекает все утверждения из JWT;
- `generateToken(extraClaims, userDetails)` – генерирует JWT с переданными дополнительными утверждениями `extraClaims` для пользователя `userDetails`.

Для работы с JWT используется библиотека JSON Web Token (`io.jsonwebtoken`) и алгоритм подписи HS256 (используется ключ, заданный в поле `SECRET_KEY`).

3.3.11 Класс `AuthenticationService`

Этот класс представляет собой сервис, который предоставляет функциональность регистрации и аутентификации пользователей в системе.

Аннотация `@Service` указывает, что этот класс является сервисом и должен быть управляемым Spring контейнером.

Класс имеет четыре поля, приведенных ниже:

- `private final UserRepo userRepo;`
- `private final PasswordEncoder passwordEncoder;`
- `private final JwtService jwtService;`
- `private final AuthenticationManager authenticationManager.`

Методы класса:

- `register()` выполняет регистрацию нового пользователя в системе;
- `authenticate()` выполняет аутентификацию пользователя в системе.

Таким образом, этот класс предоставляет функциональность регистрации и аутентификации пользователей в системе, используя Spring Security и JSON Web Token (JWT).

3.3.12 Перечисление Role

Данное перечисление представляет собой список возможных ролей пользователей системы, которые могут быть назначены сотрудникам. Константы ADMIN и USER определяют две роли: администратор и обычный пользователь.

3.3.13 Класс AuthenticationRequest

Этот класс представляет собой модель данных для запроса аутентификации пользователя в системе.

В данном классе используются аннотации фреймворка Lombok – @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как toString(), equals(), hashCode() и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Класс определяет структуру запроса на аутентификацию пользователя в системе и используется для передачи данных между клиентским и серверным приложениями. Имеет поля private String login и String password.

Эти поля нужны для представления данных, передаваемых для аутентификации пользователя.

3.3.14 Класс AuthenticationResponse

Этот класс представляет собой модель данных для ответа на запрос аутентификации пользователя в системе.

В данном классе используются аннотации фреймворка Lombok – @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как toString(), equals(), hashCode() и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Класс, как и класс, приведенный выше, также определяет структуру ответа на запрос аутентификации пользователя в системе и используется для передачи данных между серверным и клиентским приложениями.

Имеет всего одно поле:

- private String token.

Токен является строкой, которая используется для идентификации пользователя на сервере и доступа к защищенным ресурсам.

3.3.15 Класс RegisterRequest

Класс используется как часть процесса регистрации нового пользователя в системе. По сути, этот класс является моделью данных (data model), представляющей структуру запроса на регистрацию нового пользователя. В данном классе используются аннотации фреймворка Lombok

– @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как toString(), equals(), hashCode() и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

```
-private String firstName;  
-private String middleName;  
-private String lastName;  
-private String login;  
-private String password;  
-private String mail.
```

Поля этого класса описывают ту информацию, которую можно внести в запрос при регистрации пользователя: пароль, логин и мейл, а также фамилию, имя и отчество. Чтобы зарегистрировать пользователя сначала необходимо заполнить всю информацию в базе данных через отдел кадров.

3.3.16 Класс EditRequest

Этот класс представляет объект запроса на изменение данных пользователя. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером в процессе обновления профиля пользователя.

В данном классе используются аннотации фреймворка Lombok – @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как toString(), equals(), hashCode() и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

```
-private String firstName;  
-private String middleName;  
-private String lastName;  
-private Date birthD;  
-private String role;  
-private long workNumber;  
-private long personalNumber;  
-private String locationStreet;  
-private String cabinetOffice.
```

Поля этого класса описывают ту информацию, которую можно внести в запрос при изменении данных сотрудника: фамилию, имя и отчество, день рождения, роль, рабочий и персональный номера, рабочий адрес, и рабочий кабинет и корпус.

3.3.17 Класс `LoginRequest`

Этот класс представляет объект запроса для смены пароля пользователя в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

Содержит всего одно поле – `private String password`, которое содержит новый пароль.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы `JavaBean`, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.18 Класс `EventRequest`

Этот класс представляет объект запроса для создания нового события в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы `JavaBean`, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private String typeOfEvent`;
- `private String commentFe`;
- `private Date dateOfEvent`.

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании события: тип события, комментарий к нему и дата самого события.

3.3.19 Класс `LogStatementCreateRequest`

Этот класс представляет объект запроса для создания нового заявления для пользователя в системе. Содержит поля, которые могут быть изменены, включая поле `private String bodyDoc`, предназначенное для документа, если таковой имеется. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы `JavaBean`, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- private long idApprover;
- private String commentLs;
- private int daysSum;
- private int typeLeave;
- private Date dateLeave;
- private Date dateOfLs;
- private String bodyDoc.

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании заявления: сотрудника, который сможет подтвердить или опровергнуть заявление, комментарий, количество дней, тип заявления, дату начала действия заявления, дату создания заявления, и возможность прикрепить скан физической копии документа, если таковая имеется или если она необходима.

3.3.20 Класс `LogStatementRequest`

Этот класс представляет объект запроса для изменения статуса заявления в системе. Содержит одно поле, которое может быть изменено – `private int status`, предназначенное для указания статуса заявления. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы `JavaBean`, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.21 Класс `NoticeEventRequest`

Этот класс представляет объект запроса для создания связи события с тем сотрудником, которого нужно оповестить. Содержит одно поле, которое может быть изменено – `private long recipientId`, предназначенное для указания статуса заявления. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы `JavaBean`, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.22 Класс TaskRequest

Этот класс представляет объект запроса для создания нового задания в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как toString(), equals(), hashCode() и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- private Date dateTask;
- private String nameOfTask;
- private String commentTe;
- private long idExecutor.

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании задания: дату, суть задания, комментарий и пользователя, на которого будет назначено само задание.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Данное веб-приложение разработано по принципу REST, архитектурного стиля, который используется для создания веб-приложений и API. RESTful веб-приложения предоставляют доступ к ресурсам, представленным в формате URI, и выполняют операции с этими ресурсами с помощью стандартных методов HTTP, таких как GET, POST, PUT, DELETE.

Схема работы RESTful приложения, которую можно посмотреть на рисунке 4.1, обычно выглядит следующим образом:

1. Клиент отправляет запрос на сервер, используя HTTP методы: GET, POST, PUT, DELETE.
2. Сервер обрабатывает запрос и возвращает ответ в формате JSON, XML или другом формате.
3. Клиент обрабатывает ответ сервера и обновляет интерфейс приложения.

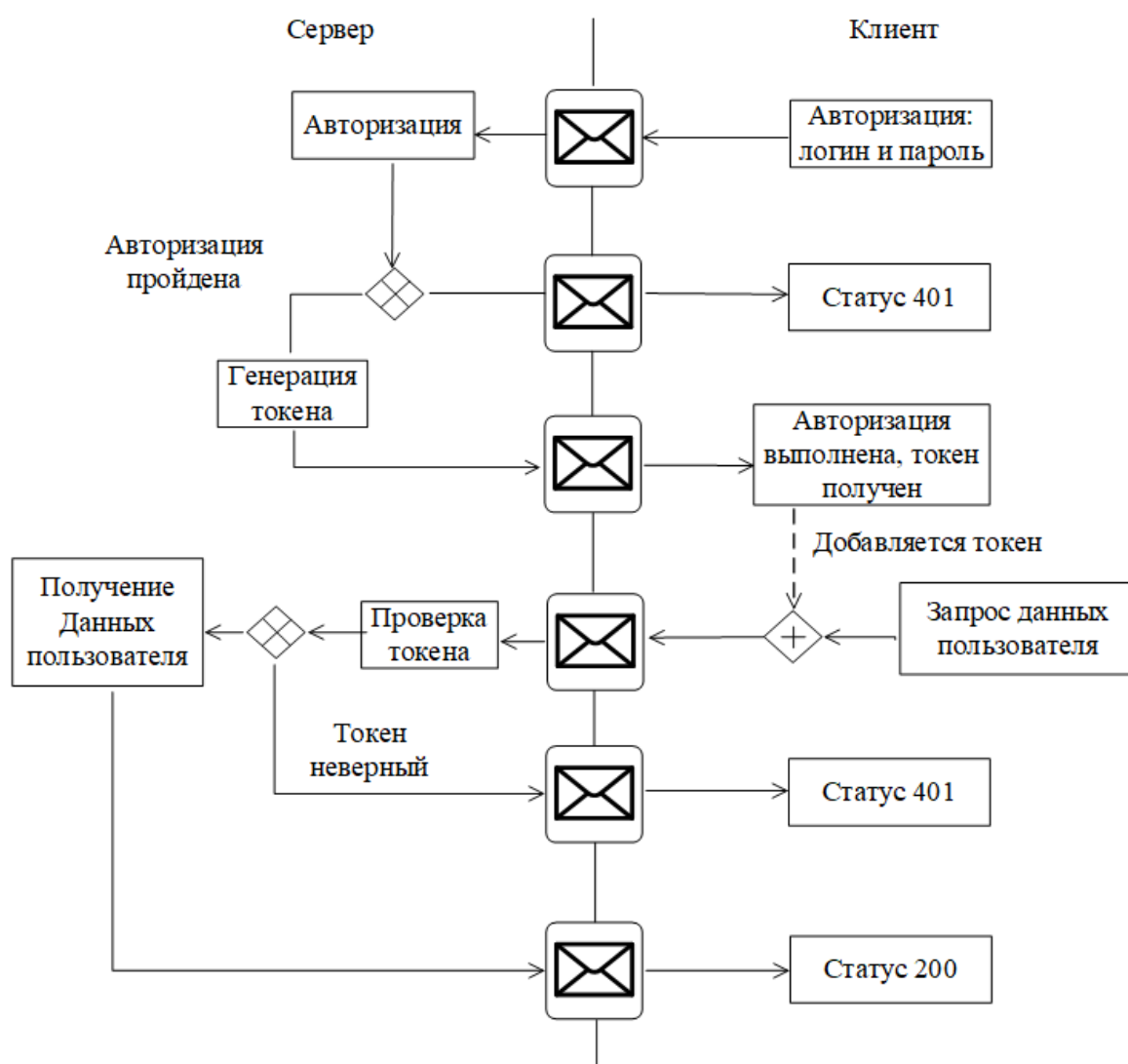


Рисунок 4.1 – Схема работы RESTful приложения

Диаграмма последовательностей для дипломного проекта на схеме ГУИР.400201.307 РР.3.

4.1 Алгоритм авторизации и регистрации

Для авторизации пользователей в данном приложении использовался JSON Web Token(JWT) – это стандарт создания токенов доступа, использующий формат JSON. Он открытый и определен в RFC 7519. Обычно, он используется для передачи данных, необходимых для аутентификации в клиент-серверных приложениях. Токены создаются на сервере, затем подписываются с помощью секретного ключа и передаются клиенту. После этого клиент использует токен для подтверждения своей личности.

Когда пользователь успешно проходит аутентификацию на сервере, сервер создает JWT и передает его обратно клиенту. JWT состоит из трех частей: заголовка, полезной нагрузки и подписи.

Весь процесс авторизации представляется следующим образом:

Пользователь отправляет запрос на аутентификацию, POST-запрос на по пути /authenticate с помощью класса AuthenticationRequest, который содержит следующие поля:

```
- private String login;  
- String password;
```

Они представляют собой логин и пароль пользователя.

Далее сервер проверяет данные с помощью класса AuthenticationService, который содержит следующий метод:

```
public AuthenticationResponse  
    authenticate(AuthenticationRequest request) {  
        authenticationManager.authenticate(new  
            UsernamePasswordAuthenticationToken(request.getLogin(),  
            request.getPassword()));  
        var user =  
userService.findUserByFirstName(request.getLogin());  
        var jwtToken = jwtService.generateToken(user);  
        return AuthenticationResponse.builder()  
            .token(jwtToken)  
            .build();  
    }
```

В этом методе происходит проверка валидности данных в authenticationManager.authenticate. Далее в методе .findUserByFirstName, с помощью базы данных, метод сверяет логин и пароль.

Пароли хранятся в зашифрованном виде с помощью хэширования, поэтому для того, чтобы сверить пароль, его необходимо сначала

раскодировать, для этого в классе `ApplicationConfig` определено поле класса `passwordEncoder`, который возвращает объект `BCryptPasswordEncoder`, который используется для хэширования пароля пользователя

Если данные прошли проверку, то метод генерирует JWT токен на основе найденного пользователя с помощью `jwtService.generateToken()`. В конце метод создаёт и возвращает объект `AuthenticationResponse`, содержащий сгенерированный токен в поле `token`. Токен содержит информацию об аутентифицированном пользователе и его правах доступа к ресурсам. В классе `JwtService` метод `generateToken()` выглядит так:

```
public String generateToken(UserDetails userDetails){
    return generateToken(new HashMap<>(), userDetails);
}
```

Для генерации токена необходим класс `UserDetails`, который содержит данные об аутентификации пользователя. Сервер отправляет сгенерированный JWT токен обратно клиенту в ответе на запрос аутентификации с помощью класса `AuthenticationResponse`, который содержит одно поле `private String token`.

При регистрации пользователя отправляется POST-запрос на `/auth/register`.

Он принимает в теле запроса объект `RegisterRequest`, содержащий данные, необходимые для регистрации нового пользователя, а именно следующие поля:

```
-private String firstName;
-private String middleName;
-private String lastName;
-private String login;
-private String password;
-private String mail.
```

Далее передает их в сервис `AuthenticationService` для выполнения регистрации с помощью метода `register`.

```
public AuthenticationResponse register(RegisterRequest
request) {
    Employee employee = employeeRepository
        .findByFirstNameAndMiddleNameAndLastName(
            request.getFirstName(),
            request.getMiddleName(),
            request.getLastName()).orElse(new
Employee());
    var user = User.builder()
```

```

        .idLogin(employee.getId())
        .firstName(request.getLogin())
        .mail(request.getMail())

.password(passwordEncoder.encode(request.getPassword()))
        .role(Role.USER)
        .build();
userService.saveUser(user);
var jwtToken = jwtService.generateToken(user);
return AuthenticationResponse.builder()
        .token(jwtToken)
        .build();
}

```

В данном методе, как и при авторизации задействовано шифрование паролей. Это значит, что при регистрации и сохранении нового пароля для пользователя система вызывает метод `passwordEncoder.encode()`, чтобы пароли хранились сразу в зашифрованном виде.

Затем он возвращает объект `AuthenticationResponse`, содержащий информацию об успешности регистрации и авторизации нового пользователя, то есть содержит поле `private String token` с токеном.

При каждом запросе, требующем авторизации, клиент добавляет JWT токен в заголовок запроса `Authorization`. Например: `Authorization: Bearer <JWT_TOKEN>/`

Если токен истек или был поврежден, сервер возвращает ошибку аутентификации. Клиент в таком случае может запросить новый токен, повторив процесс аутентификации снова.

Для работы с токеном используется класс `JwtService`, содержит ряд методов:

1. Метод `extractUsername` извлекает имя пользователя из токена.
2. Метод `extractClaim` позволяет извлекать другие поля из токена, используя функцию-резольвер.
3. Метод `generateToken` генерирует токен, используя переданные дополнительные поля и данные пользователя. Токен подписывается с помощью ключа HMAC SHA-256.
4. Метод `isTokenValid` проверяет, действителен ли переданный токен и соответствует ли он данным пользователя.
5. Метод `isTokenExpired` проверяет, истек ли срок действия токена.
6. Метод `extractExpiration` извлекает дату истечения срока действия токена.
7. Метод `extractAllClaims` извлекает все поля токена.
8. Метод `getSignInKey` возвращает ключ для подписи токена. Ключ загружается из свойства `SECRET_KEY`, которое содержит закодированный ключ в формате Base64.

```

        private static final String SECRET_KEY =
"77217A25432A462D4A614E645267556B58703273357538782F413F4428472B4
B";

        public String extractUsername(String jwtToken) {
            return extractClaim(jwtToken, Claims::getSubject);
        }

        public <T> T extractClaim(String jwtToken,
Function<Claims, T> claimsResolver){
            final Claims claims = extractAllClaims(jwtToken);
            return claimsResolver.apply(claims);
        }

        public String generateToken(
            Map<String, Object> extraClaims,
            UserDetails userDetails
        ){
            return Jwts
                .builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new
Date(System.currentTimeMillis()))
                .setExpiration(new
Date(System.currentTimeMillis() + 1200 * 60 * 24))
                .signWith(getSignInKey(),
SignatureAlgorithm.HS256)
                .compact();
        }

        public String generateToken(UserDetails userDetails){
            return generateToken(new HashMap<>(), userDetails);
        }

        public boolean isTokenValid (String jwtToken, UserDetails
userDetails){
            final String username = extractUsername(jwtToken);
            return (username.equals(userDetails.getUsername()))
&&
                !isTokenExpired(jwtToken);
        }

        private boolean isTokenExpired(String jwtToken) {
            return extractExpiration(jwtToken).before(new
Date());
        }

        private Date extractExpiration(String jwtToken) {
            return extractClaim(jwtToken, Claims::getExpiration);
        }

```

```

private Claims extractAllClaims(String jwtToken) {
    return Jwts
        .parserBuilder()
        .setSigningKey(getSignInKey())
        .build()
        .parseClaimsJws(jwtToken)
        .getBody();
}

private Key getSignInKey() {
    byte[] keyBytes =
Decoders.BASE64.decode(SECRET_KEY);
    return Keys.hmacShaKeyFor(keyBytes);
}

```

4.2 Работа Spring Security

Spring Security – это фреймворк для обеспечения безопасности в приложениях на платформе Spring. Он предоставляет широкий спектр функций для аутентификации, авторизации и защиты от атак, таких как CSRF, XSS и других.

Именно он позволяет шифровать пароли с помощью класса `UserDetails`, с использованием `BCrypt`. С его же помощью он позволяет создавать данные о авторизированном пользователе, что позволяет использовать это для работы с базой данных и получении данных о пользователе.

С помощью метода `securityFilterChain()` в классе `SecurityConfig` создается цепочка фильтров безопасности, которые позволяют управлять сессиями. Она определяет порядок, в котором фильтры будут применяться, и какие запросы будут обрабатываться каждым фильтром. Этот метод создает цепочку фильтров для обработки запросов HTTP. Метод использует объект `HttpSecurity`, который предоставляет DSL (Domain-Specific Language) для настройки конфигурации Spring Security.

```

public SecurityFilterChain securityFilterChain(HttpSecurity
httpSecurity) throws Exception {
    httpSecurity
        .cors(Customizer.withDefaults())
        .and()
        .addFilterBefore(new
CorsFilter(corsConfigurationSource()),
SessionManagementFilter.class)
        .httpBasic().disable()
        .csrf().disable()
        .authorizeHttpRequests()
        .requestMatchers("/**")
        .permitAll()

```

```

        .anyRequest()
        .authenticated()
        .and()
        .sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()

.authenticationProvider(authenticationProvider)
        .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
        return httpSecurity.build();
    }

```

Также в этом классе есть метод `CorsConfigurationSource`, который определяет, какие запросы от каких источников и с какими методами HTTP разрешены для взаимодействия с веб-приложением.

В методе создается объект `CorsConfiguration`, который содержит информацию о разрешенных origins и методах, а также другие настройки, такие как заголовки CORS (например, «Access-Control-Allow-Origin»). Затем определяется источник конфигурации на основе URL-адреса источника и конфигурации CORS, и возвращается созданный объект `CorsConfigurationSource`.

```

    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new
CorsConfiguration();

configuration.setAllowedOrigins(List.of("http://localhost:3000")
);

configuration.setAllowedMethods(Arrays.asList("GET", "POST"));
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**",
configuration);
        return source;
    }

```

Эти же методы связаны с классом `JwtAuthenticationFilter`, который содержит один метод `doFilterInternal()`. Когда клиент отправляет запрос на сервер, он проходит через этот фильтр. В методе `doFilterInternal()`, он проверяет заголовок `Authorization` запроса, чтобы определить, есть ли токен в запросе. Если заголовок отсутствует или токен отсутствует, то фильтр пропускает запрос к следующему фильтру в цепочке.


```

        protected void doFilterInternal(@NonNull HttpServletRequest
request, @NonNull HttpServletResponse response, @NonNull
FilterChain filterChain) throws ServletException, IOException {
            final String authHeader =
request.getHeader("Authorization");
            final String jwtToken;
            final String login;
            if (authHeader == null ||
!authHeader.startsWith("Bearer")) {
                filterChain.doFilter(request, response);
                return;
            }
            jwtToken = authHeader.substring(7);
            login = jwtService.extractUsername(jwtToken);
            if(login != null &&

SecurityContextHolder.getContext().getAuthentication() == null){
                UserDetails userDetails =
this.userDetailsService.loadUserByUsername(login);
                if(jwtService.isTokenValid(jwtToken,
userDetails)){
                    UsernamePasswordAuthenticationToken
authenticationToken =
                        new
UsernamePasswordAuthenticationToken(
                            userDetails,
                            null,

userDetails.getAuthorities());
                    authenticationToken.setDetails(
                        new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authenticat
ionToken);
                }
            }
            filterChain.doFilter(request, response);
        }

```

Если в запросе есть токен, то фильтр извлекает логин пользователя из токена, затем проверяет, не была ли аутентификация пользователя уже выполнена в текущем контексте безопасности. Если пользователя еще не было аутентифицировано, фильтр получает детали пользователя из `userDetailsService` и проверяет токен на его действительность с помощью `jwtService.isTokenValid()`. Если токен действителен, то создается `UsernamePasswordAuthenticationToken`, содержащий `UserDetails`, и устанавливается в текущий контекст безопасности через `SecurityContextHolder.getContext().setAuthentication(authenticationToken)`.

Фильтр затем передает запрос следующему фильтру в цепочке с помощью `filterChain.doFilter(request, response)`. Если пользователь был успешно аутентифицирован, то его контекст безопасности будет доступен для последующих запросов в рамках этой сессии.

4.3 Алгоритм запросов

После того, как пользователь регистрируется, получит токен на время сессии он может осуществлять запросы к серверу, чтобы взаимодействовать с сервером.

При каждом запросе будет срабатывать `SecurityFilter`, рассмотренный в разделе.

Для того, чтобы пользователь получил данные, с клиента отправляется GET или POST-запрос.

Например, чтобы получить данные и главную страницу о авторизованном сотруднике в классе `MainPageController` вызывается метод `getEmployeeInfo(Authentication authentication)`

```
@GetMapping(path = "/mainUserInfo", produces =
MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<List<EmployeeFullView>>
getEmployeeInfo(Authentication authentication) {
    return
    ResponseEntity.ok(employeeFullViewService.findAllByLoginUser
        (authentication.getName()));
}
```

Этот метод является обработчиком GET-запроса на путь «`/mainUserInfo`» и возвращает список информации о сотрудниках в формате JSON. В качестве параметра метод принимает объект `Authentication`, который содержит информацию об аутентифицированном пользователе.

Метод использует эту информацию для получения имени пользователя из объекта `Authentication` и передачи его в качестве аргумента в метод `employeeFullViewService.findAllByLoginUser()`, который возвращает список `EmployeeFullView`, содержащих информацию о сотрудниках.

Метод `findAllByLoginUser()` работает через класс `EmployeeFullViewService`, который связан с интерфейсом `EmployeeFullViewRepository` с расширением `JpaRepository<EmployeeFullView, Long>`.

В результате успешного выполнения метод возвращает HTTP-ответ со статусом 200 (OK) и телом ответа, содержащим список информации о сотрудниках в формате JSON.

Кроме информации о сотруднике на главной странице содержится информация о событиях, заявлениях и заданиях, все эти методы являются GET-запросами. Их можно рассмотреть далее.

Запрос о просмотре заявлений выглядит так:

```
@GetMapping(path = "/ls", produces =
MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<List<LogStatementsView>>
getLsRequests(Authentication authentication) {
    return
ResponseEntity.ok(this.logStatementsViewService.findAllByIdAppro
verAndStatus(

userService.findUserByFirstName(authentication.getName()).getIdL
ogin(), 3));
}
```

Этот запрос о событиях авторизованного пользователя:

```
@GetMapping(path = "/events", produces =
MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody List<EventsView>
getEvents(Authentication authentication) {
    return this.eventsViewService.findAllByIdRecipient(

userService.findUserByFirstName(authentication.getName()).getIdL
ogin());
}
```

Этот запрос о задачах авторизованного пользователя:

```
@GetMapping(path = "/tasks", produces =
MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody List<TasksView>
getTasks(Authentication authentication) {
    System.out.println(authentication);
    return this.tasksViewService.findAllByIdExecutor(

userService.findUserByFirstName(authentication.getName()).getIdL
ogin());
}
```

Все эти три запроса основаны на извлечении данных о пользователе с помощью объекта `Authentication`, переданного в качестве параметра.

На главной страничке предусмотрена возможность подтверждать или опровергать заявления, это происходит через POST-запрос `setLsApprove()`. Для него необходимо знать номер заявления. Он передается с помощью кода клиентской части, от пользователя будет необходимо только нажать кнопку.

```
@PostMapping(path = "/ls/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<String> setLsApprove(
```

```

        @PathVariable Long id,
        Authentication authentication,
        @RequestBody LogStatementRequest request
    ) {
        LogStatement logStatement =
logStatementService.findByIdAndIdApprover(
            id,

userService.findUserByFirstName(authentication.getName()).getIdL
ogin()

                ).orElse(new LogStatement());
        logStatement.setStatus(request.getStatus());
        System.out.println(logStatement.getId() + "    " +
            logStatement.getStatus());

if(logStatementService.saveLogStatement(logStatement))
    System.out.println("ok");
    return ResponseEntity.ok("Done ");
    }

```

Принцип создания заявлений, событий и заданий похож. Все они представляют собой POST-запросы, используя классы-сервиса, в которые с помощью классов-запросов с содержанием полей вносятся данные пользователя, потом проверяются на валидность и сохраняются в базу данных. Пример подобного запроса:

```

    @PostMapping(path = "/lscreate", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> createLS( @RequestBody
LogStatementCreateRequest request,

                                                Authentication
authentication

                                                ){
        LogStatement logStatement = LogStatement.builder()
            .id(0)
            .idApprover(request.getIdApprover())
            .status(3)

            .idEmployee(userService.findUserByFirstName(authentication.getNa
me()).getIdLogin())

                .commentLs(request.getCommentLs())
                .typeLeave(request.getTypeLeave())
                .dateLeave(request.getDateLeave())
                .dateOfLs(request.getDateOfLs())
                .daysSum(request.getDaysSum())
                .build();

if(logStatementService.saveLogStatementAll(logStatement))
    System.out.println("ok");
    if(request.getBodyDoc() != null) {

```

```

        Document document = Document.builder()
            .bodyDoc(request.getBodyDoc())
            .idIs(logStatement.getId())
            .build();
        if (documentService.saveDocument(document))
            System.out.println("ok");
    }
    return ResponseEntity.ok("Done ");
}

```

В данном методе идет сохранение в две таблицы LogStatement и Document в силу того, что они связаны друг с другом. К заявлению можно приложить документ, его физическую копию, если таковой имеется.

Запрос на смену представляет собой POST-запрос, который через класс LoginRequest, который также вызывает класс для шифрования passwordEncoder пароля.

```

    @PostMapping(path = "/password", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> changePassword(@RequestBody
LoginRequest request,
                                                    Authentication
authentication) {
        User user =
userService.findUserByFirstName(authentication.getName());

user.setPassword(passwordEncoder.encode(request.getPassword()));
        if (userService.saveUserPassword(user))
            System.out.println("ok");
        return ResponseEntity.ok("Done");
    }

```

Метод обрабатывает GET-запрос на получение списка сотрудников в виде JSON. Он вызывает метод findAll() из employeesViewService для получения списка всех сотрудников, создает ResponseEntity с HTTP-статусом "200 OK" и списком сотрудников в теле ответа, и возвращает его в качестве ответа на запрос.

```

    @GetMapping(path = "/employees", produces =
MediaType.APPLICATION_JSON_VALUE)
    public @ResponseBody List<EmployeesView> getEmployees() {
        return this.employeesViewService.findAll();
    }

```

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Для разработки системы использовался такой подход к тестированию, как Test-driven development (TDD). Методология разработки программного обеспечения, при которой разработчик пишет тесты для каждой функции или компонента программы перед тем, как реализовать его код. Эта методология пропагандирует написание тестов перед написанием кода, в отличие от классического подхода к разработке, когда сначала пишется код, а затем тесты на его проверку.

Ближе к концу разработки проводится рефакторинг нового кода для приведения его к соответствующим стандартам. Такой подход является полной противоположностью разработке, при которой сначала разрабатывается программное обеспечение, а затем описываются тестовые ситуации.

Основная идея TDD заключается в том, что писание тестов перед кодом позволяет программисту более точно определить, как должна работать программа, а также позволяет быстро выявлять ошибки и проблемы в процессе разработки. Это помогает повысить качество кода и ускоряет процесс разработки.

В программировании тестирование – это процесс, который определяет, работает ли написанный код правильно. Обычно программисты выполняют тестирование функционала вручную, стимулируя код и проверяя его выполнение. Однако, при автоматическом тестировании компьютер сам стимулирует код и проверяет его работоспособность, что позволяет программисту увидеть результат на дисплее. Это приводит к изменению ответственности, так как теперь от тестов зависит соответствие кода техническому заданию. Методика разработки через тестирование заключается в создании автоматических тестов.

Разработка через тестирование - это методология, в которой разработчик создает автоматизированные модульные тесты, которые определяют требования к коду перед его написанием. Тесты содержат проверки условий, которые могут либо выполниться, либо нет. Когда все условия выполняются, тест считается пройденным, и это подтверждает поведение, которое разработчик ожидает от своего кода. Для создания и автоматизации запуска тестов разработчики используют библиотеки для тестирования. Обычно модульные тесты покрывают критические и нетривиальные участки кода, которые могут быть подвержены частым изменениям, от которых зависит работоспособность другого кода, или которые имеют много зависимостей.

Среда разработки должна быстро реагировать на небольшие модификации кода. Архитектура программы должна базироваться на использовании множества сильно связанных компонентов, которые слабо соединены друг с другом, благодаря чему тестирование кода упрощается.

Техника TDD не только предназначена для проверки правильности работы кода, но также оказывает влияние на дизайн программы. Опираясь на

написанные тесты, разработчики могут лучше понять, какая функциональность необходима для пользователей. Таким образом, детали интерфейса могут быть определены на ранней стадии разработки, до полной реализации решения. Тесты также должны соответствовать стандартам кодирования, применяемым к основному коду.

На рисунке 5.1 приведена схема разработки с применением TDD.

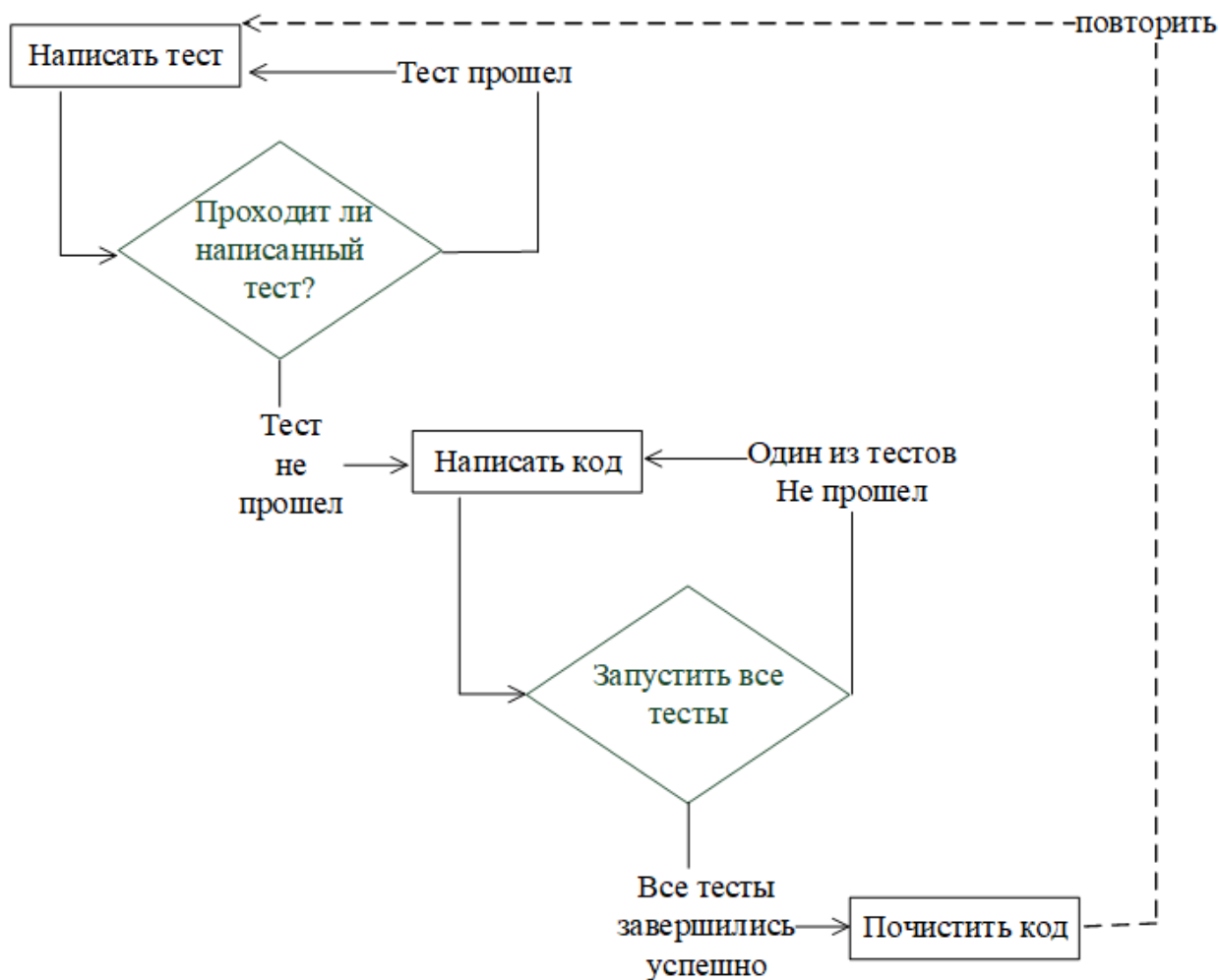


Рисунок 5.1 – Разработка с использованием TDD

В современных проектах значительное внимание уделяется созданию автоматических тестов, поскольку они являются важной частью процесса разработки программного обеспечения. Стандарты разработки, такие как TDD (Test Driven Development) и BDD (Behaviour Driven Development), часто используются для обеспечения высокого покрытия автоматическими тестами. BDD является одним из подходов TDD и может быть реализован в системе RSpec.

RSpec - это фреймворк для написания автоматических тестов на языке программирования Ruby, который предоставляет специальный язык программирования (DSL), называемый спецификацией, для написания тестов.

RSpec используется для практики BDD, которая позволяет описывать желаемое поведение системы на языке бизнес-задач. Спецификация - это отдельный файл, содержащий описание определенной части программы, такой как контроллер, модель, шаблон или хелпер. Файлы спецификаций должны храниться в специальной поддиректории проекта с названием "spec", а имена файлов должны заканчиваться на "_spec.rb".

Иногда возникают ситуации, когда внесение изменений или добавление новых функций может непреднамеренно повлиять на уже существующую функциональность, и разработчик может не заметить эти изменения. Это явление называется регрессией.

Регрессионное тестирование относится к категории тестирования программного обеспечения, которое направлено на выявление возможных ошибок в уже протестированном функционале. Оно используется для обнаружения регрессионных ошибок, которые могут возникнуть при добавлении новых функций или исправлении ошибок, что может привести к появлению новых ошибок в ранее протестированных частях программного кода. Регрессионное тестирование не является средством для доказательства отсутствия ошибок в программе.

Для регрессионного тестирования используются методы, которые включают повторное выполнение предыдущих тестов с целью обнаружения новых регрессионных ошибок, которые могут появиться при добавлении нового функционала. В процессе регрессионного тестирования проводится верификация устранения новых дефектов, проверка того, что дефект, который был исправлен ранее, не возникает снова, и проверка того, что функциональность приложения не нарушена после добавления нового кода и исправления дефектов.

Выделяют несколько видов регрессионных тестов:

- верификационные тесты (проводятся для проверки исправления, обнаруженной ранее ошибки);
- тестирование верификации версии (проверка работоспособности основной функциональности программы).

При написании исходного кода веб-приложения тестирование проводилась в три этапа:

- тестирование каждого отдельного блока в процессе написания программного кода;
- тестирование взаимодействия нескольких блоков между собой;
- полное тестирование программы после окончания процесса написания программного кода.

Эти этапы тестирования являются взаимосвязанными и необходимыми. Модульное тестирование необходимо для выявления проблем в отдельных компонентах программы, а анализ работы программы в целом может быть затруднительным без этого этапа. Однако, работоспособность каждого компонента в отдельности не гарантирует корректного поведения всей программы.

При разработке веб-приложения было проведено модульное тестирование, чтобы выявить возможные проблемы в отдельных компонентах приложения. Также важно отметить проведение сквозного тестирования, которое позволяет проверить работу приложения в целом и выявить возможные проблемы, связанные с взаимодействием различных компонентов приложения.

5.1 Модульное тестирование

Для тестирования серверной части использовалось модульное тестирование. Под подобным тестированием подразумевается процесс, позволяющий проверить отдельные или методы исходного кода.

Основная цель модульного тестирования заключается в том, чтобы писать тесты для каждой функции или метода и проверять все возможные сценарии, к которым можно прийти, как позитивные, так и негативные. Благодаря подобному подходу можно достаточно быстро проверять уже протестированные части кода на регрессии и отслеживать не привело ли последующее изменение к ней. Такой подход облегчает обнаруживать и исправлять подобные ошибки.

5.2 Сквозное тестирование

Сквозное тестирование (end-to-end testing) – это вид тестирования, который проводится на уровне системы в целом. Оно используется для проверки работы программного продукта в условиях, максимально приближенных к реальным. В отличие от модульного тестирования, где проверяется отдельный модуль кода, сквозное тестирование проверяет взаимодействие между компонентами системы в целом.

При сквозном тестировании проводится проверка функциональности всей системы в целом, начиная от пользовательского интерфейса и заканчивая базой данных и взаимодействием с другими системами. На каждом этапе производится проверка входных и выходных данных, а также корректности и своевременности выполнения действий системы.

Сквозное тестирование позволяет выявить не только ошибки в отдельных модулях, но и проблемы взаимодействия между ними. Это может включать ошибки конфигурации, проблемы совместимости, ошибки при передаче данных и другие проблемы, которые могут возникнуть только при взаимодействии компонентов системы в целом.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Требования к аппаратному обеспечению

Минимальные требования для работы приложения могут быть определены по минимальным требованиям, необходимым для запуска современного веб-браузера. Самым популярным браузером является Google Chrome, поэтому оптимальным решением будет взять требования к аппаратному обеспечению исходя из его требований.

Требования к аппаратному обеспечению клиентской части приложения представлены в таблице 6.1.

Таблица 6.1 – Требования клиентской части к аппаратному обеспечению

	Windows	Mac	Linux
Операционная система	Windows 7 Windows 8 Windows 8.1 Windows 10	Mac OS X 10.10	Ubuntu 14.4 Debian 8 OpenSUSE 13.3 Fedora Linux 14
Процессор	Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2		
Видеоадаптер	3D адаптер nVidia, Intel, AMD/ATI		
Видеопамять	64 Мб		
Свободное место на диске	350 Мб		
Оперативная память	512 Мб		

Минимальные требования к аппаратному обеспечению компьютера для развертывания серверной части:

1. Процессор Intel Core i5 7400.
2. Объем свободного пространства на постоянном запоминающем устройстве – пятьдесят ГБ файлового хранилища.
3. Объем оперативного запоминающего устройства – шестнадцать ГБ.
4. Наличие доступа к сети Internet.

Конкретные требования к аппаратному обеспечению могут измениться в зависимости от объема данных, которые приложение будет обрабатывать, и количества пользователей, которые им будут пользоваться.

6.2 Руководство по развертыванию приложения

Разработанный проект написан на языке программирования Java с использованием фреймворка Spring для серверной части, библиотека React для клиентской части. В нашем случае приложение изначально разрабатывалось на Windows 10,

Для запуска серверной части в первую очередь необходимо установить Java Development Kit(JDK):

1. Скачать установочный файл JDK с официального сайта Java – Oracle и скачать JDK для своей операционной системы.

2. Запустить установочный файл.

3. Выбрать директорию установки JDK.

4. Настроить переменную окружения PATH.

Далее надо настроить базу данных. Для базы данных будет использоваться PostgreSQL. Для установки серверной и клиентской части PostgreSQL необходимо скачать установщик с официального сайта, а далее, следуя его советам, установить базу данных на устройство.

Для запуска клиентской части приложения необходимо скачать и установить последнюю версию Node.js – это среда выполнения JavaScript, которая позволяет запускать JavaScript-код на сервере. Для этого необходимо скачать его с официального сайта.

Далее надо установить все сторонние пакеты зависимостей, перечисленные в файле package.json, необходимые для работы приложения. Для этого необходимо выполнить команду:

```
npm install
```

Далее идет развертывание с помощью команды

```
npm run build
```

Команда создает каталог сборки с производственной сборкой приложения. После этого необходимо настроить HTTP-сервер таким образом, чтобы посетитель вашего сайта получал index.html, а запросы к статическим путям, например /static/js/main.<hash>.js, получали содержимое файла /static/js/main.<hash>.js.

Чтобы установить статический сервер для окружений, использующих Node, для облегчения его настройки будет установка библиотеки serve и предоставление ей возможности сделать все остальное:

```
npm install -g serve  
serve -s build
```

Последняя команда, показанная выше, будет обслуживать статический сайт на порту 3000. Как и многие другие внутренние настройки serve, порт можно настроить с помощью флагов -l или --listen:

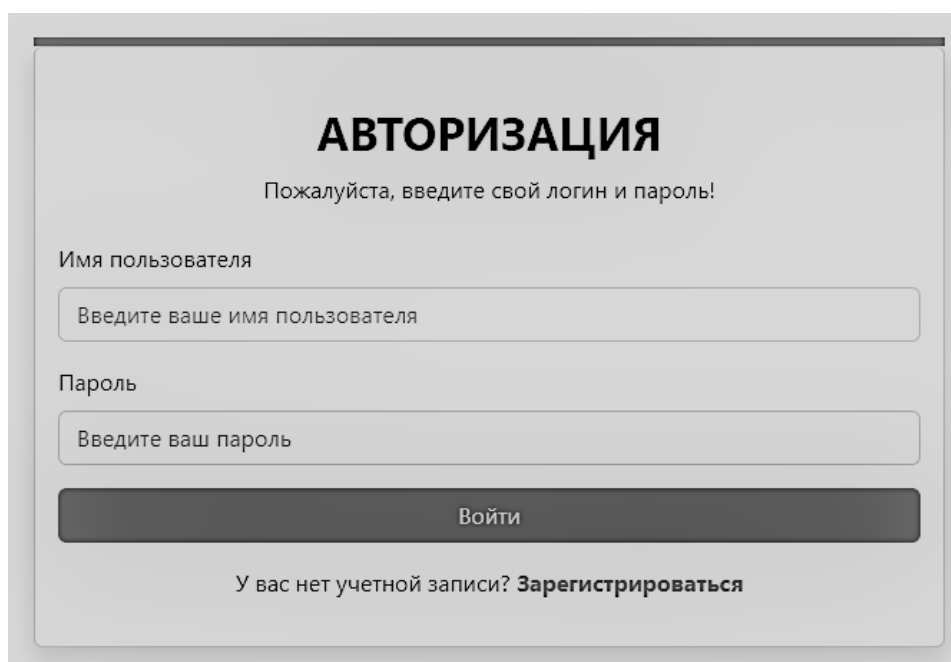
```
serve -s build -l 4000
```

Основная настройка серверной части завершена. Для локального запуска приложения необходимо выполнить настройку с помощью команды `mvn clean install`, запустить серверную часть с помощью команды `java -jar` и запустить клиентскую часть с помощью команды `npm start`.

6.3 Руководство по использованию программного обеспечения

6.3.1 Авторизация и регистрация

Первая страница, на которую будет перенаправлен пользователь, будет страница авторизации, представленная на рисунке 6.1.



The image shows a web form for authentication. At the top, the title "АВТОРИЗАЦИЯ" is centered in a large, bold font. Below it, a subtitle "Пожалуйста, введите свой логин и пароль!" is centered. The form consists of two input fields: the first is labeled "Имя пользователя" and contains the placeholder text "Введите ваше имя пользователя"; the second is labeled "Пароль" and contains the placeholder text "Введите ваш пароль". Below these fields is a dark gray button with the text "Войти" in white. At the bottom of the form, there is a link that says "У вас нет учетной записи? [Зарегистрироваться](#)".

Рисунок 6.1 – Авторизация

Здесь пользователь видит поля для заполнения, а именно логин, он же имя пользователя, и пароль, которые являются уникальными для каждого сотрудника. Если поля не будут введены корректно, то они будут обведены красным, давая понять, что есть проблема.

Если у пользователя нет аккаунта, то у него есть возможность создать свой кабинет самостоятельно. Для этого есть кнопка «Зарегистрироваться», которая перенаправит пользователя на новую страницу, где ему для регистрации пользователю необходимо заполнить следующие поля:

- фамилия;
- имя;
- отчество;
- логин;
- личная электронная почта;
- пароль.

Данную страницу можно посмотреть на рисунке 6.2.

РЕГИСТРАЦИЯ
Пожалуйста, введите свои личные данные!

Фамилия:

Имя:

Отчество:

Имя пользователя:

Электронная почта:

Пароль:


Рисунок 6.2 – Регистрация

Перед регистрацией пользователя отделу кадров необходимо заполнить информацию о сотруднике, которую он подает при устройстве на работу. Кроме этого сотруднику должны дать инструкции о безопасном составлении пароля: каким образом сгенерировать безопасный пароль, где и как его можно хранить, а также через какой период времени его необходимо менять на новый.

6.3.2 Главная страница

После авторизации или регистрации пользователь переходит на свою личную страницу, представленную на рисунке 6.3.

Главная Страница | Пользователи



Рябов Игнатий Викторович
#1243423

Дата рождения: 12.05.1996
Рабочий адрес: 690 Waiwaka Light Apt. 579
Номер телефона: +375 (25) 453-12-56
Рабочий номер телефона: +375 (25) 983-72-86

Электронная почта: khrop1725@andorem.com
Отдел: Пресс-служба
Должность: Декан
Кабинет: 5109

события

Событие #1123213

Собрание кафедры в аудитории 511/5 в 16.00
25.05.2023

задачи

заявления

Рисунок 6.3 – Главная страница, события

На ней он может увидеть отображение всех данных о себе, кнопку смены пароля, а также три вида оповещений: события, задачи и заявления. На верхней панели отображены три кнопки:

- главная страница;
- пользователи;
- значок выхода из системы.

Информация о себе представлена в виде карточки, на которой отображены поля и с информацией. Под ней три кнопки:

- события;
- задачи;
- заявления.

Под кнопками отображаются три раздела. В каждом разделе в виде отдельных карточек будут представлена информация о разделе, и тематическая кнопка, которая позволит создавать похожие информационные карточки для других сотрудников.

На рисунке 6.3 можно увидеть пример событий: карточка с событием, которые предназначена для авторизованного сотрудника и кнопка создания события, которая позволит создавать их и назначать на других людей. Пример создания событий можно увидеть на рисунке 6.4.

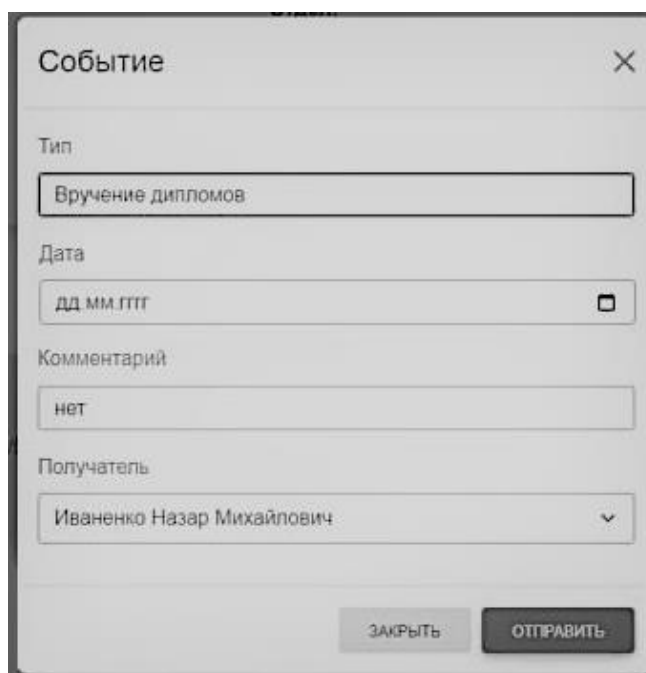


Рисунок 6.4 – Создание события

Для создания события предоставлено четыре поля:

- тип события;
- дата события;
- комментарий к событию;
- получатель, которому предназначено событие.

После заполнения всех важных полей можно подтвердить отправку события пользователю.

Создание событий, как и заявлений и заданий представляют собой всплывающие окна.

Далее на рисунке 6.5 рассмотрим поле задач.

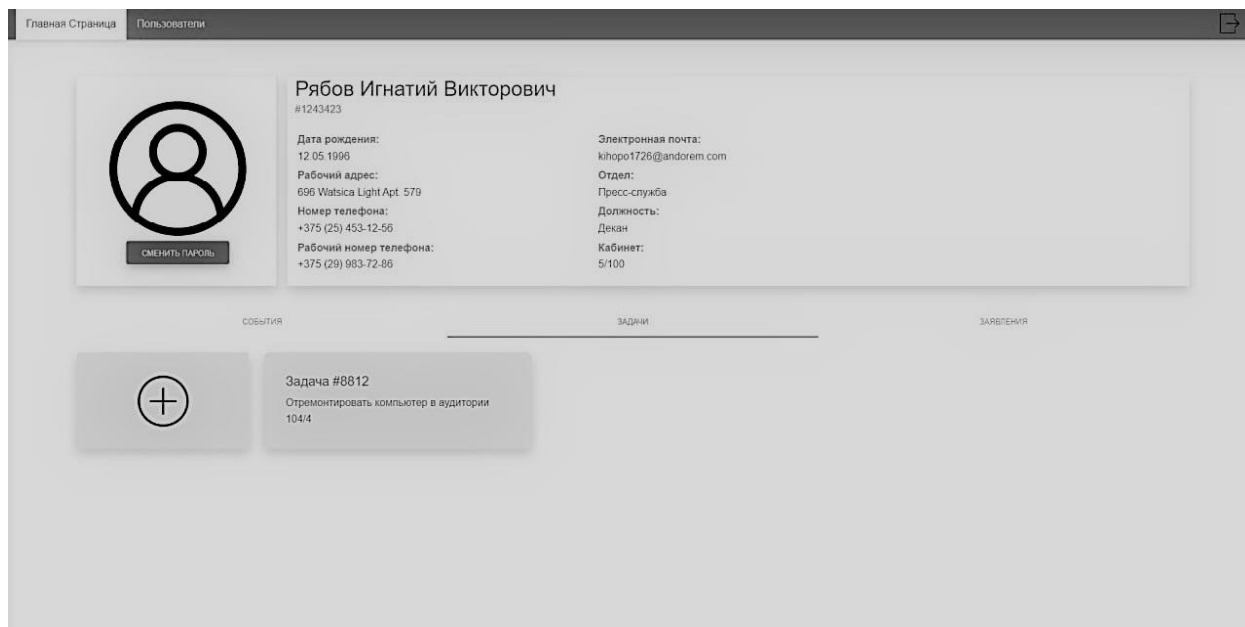


Рисунок 6.5 – Главная страница, задачи

Как и с событиями, точно таким же образом можно создавать и задачи, пример интерфейса представлен на рисунке 6.6.

The screenshot shows a modal dialog box titled 'Задача' (Task). It contains three input fields: 'Название' (Name) with the placeholder text 'Введите название', 'Комментарий' (Comment) with the placeholder text 'Введите комментарий', and 'Исполнитель' (Executor) with a dropdown menu showing 'Выберите исполнителя'. At the bottom of the dialog, there are two buttons: 'ЗАКРЫТЬ' (Close) and 'ОТПРАВИТЬ' (Send).

Рисунок 6.6 – Создание задачи

Для создания задачи представлено три поля:
- суть задачи;

- комментарий к задаче;
- назначение на человека, которому предназначена задача.

После заполнения всех важных полей можно подтвердить отправку задачи пользователю.

Последнее, что можно сделать на главной странице – это открыть раздел с заявлениями. В заявлениях по похожему принципу будут отображаться карточки с заявлениями, которые предназначены авторизованному пользователю, но с одним отличием от задач и событий. Заявления нуждаются в подтверждении или опровержении, которые отображены двумя кнопками, интерфейс которых можно посмотреть на рисунке 6.7.

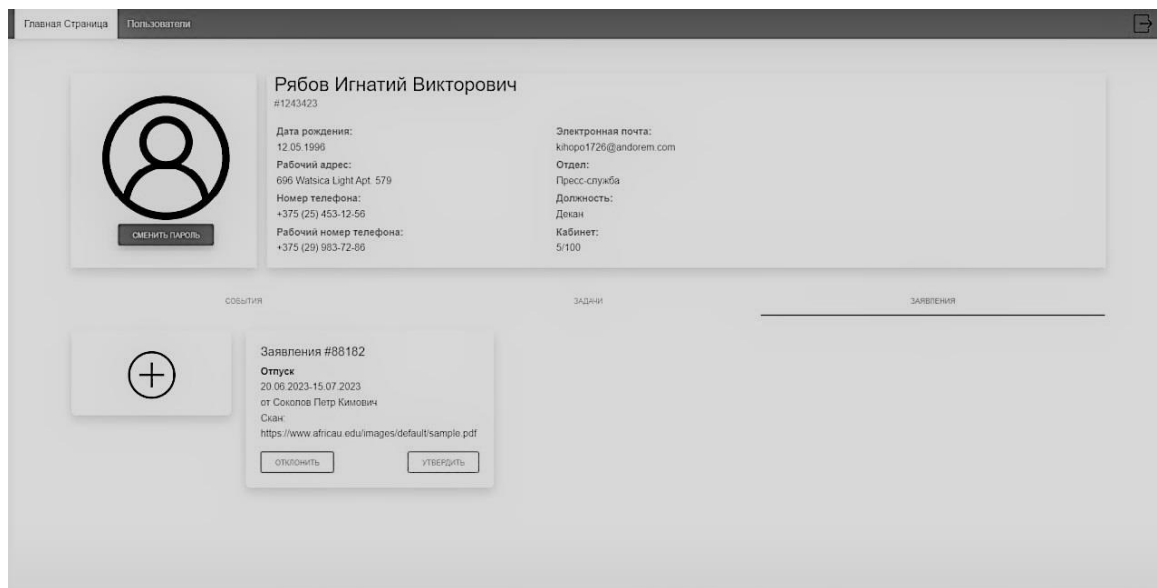


Рисунок 6.7 – Главная страница, заявления

The form titled 'Заявление' (Statement) contains the following fields:

- Тип заявления** (Statement type): A dropdown menu currently showing 'Причина отсутствия' (Reason for absence).
- Дата ухода** (Date of departure): A date picker showing 'дд.мм.гггг'.
- Срок отсутствия** (Duration of absence): A text input field with the placeholder 'Введите срок отсутствия'.
- Комментарий** (Comment): A text input field with the placeholder 'Введите комментарий'.
- Руководитель** (Manager): A dropdown menu with the placeholder 'Выберите руководителя'.
- Ссылка на документ** (Document link): A text input field.

 At the bottom right, there are two buttons: 'ЗАКРЫТЬ' (Close) and 'ОТПРАВИТЬ' (Send).

Рисунок 6.8 – Создание заявления

На рисунке 6.8, представленном страницей ранее, можно увидеть интерфейс создания заявления, по схожему с созданием заявлений и задач принципу. Имеет шесть полей:

- тип заявления;
- дата ухода (начала действия заявления);
- срок отсутствия работника, если таковой необходим;
- комментарий к заявлению;
- руководитель, которому данное заявление отправляется;
- ссылка на скан физического документа.

После заполнения всех полей можно отправить заявление руководителю.

6.3.3 Пользователи

На главной странице есть кнопка «Пользователи», которая перенаправляет на страницу со списком всех сотрудников.

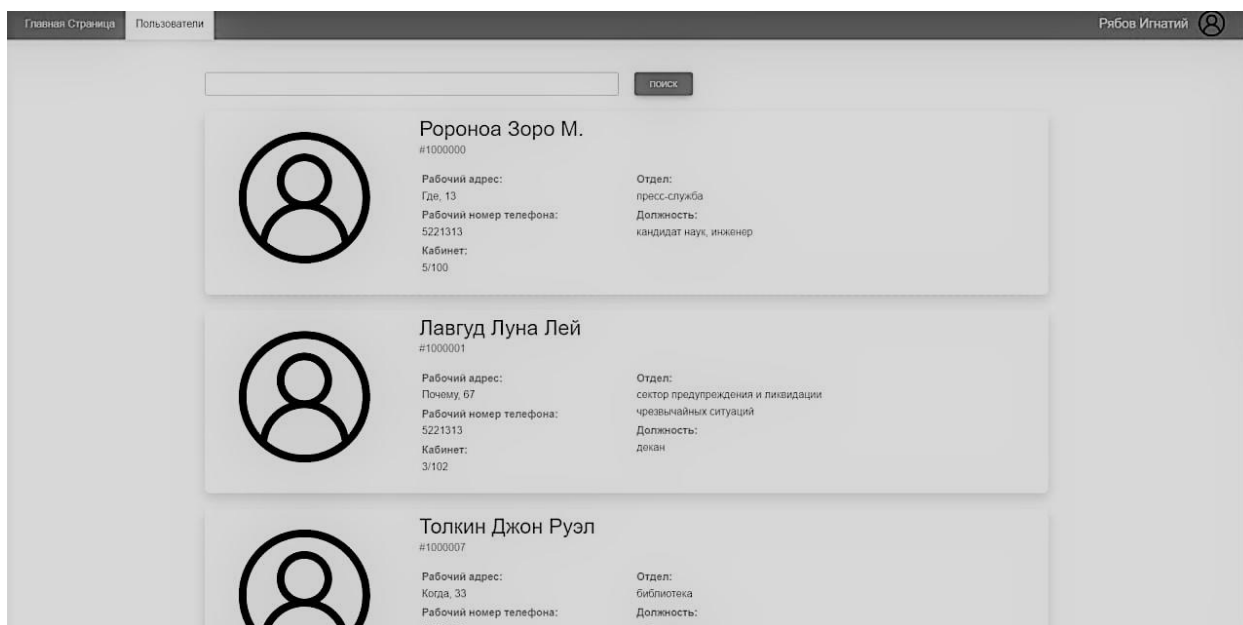


Рисунок 6.9 – Список сотрудников

Открыв всех пользователей, можно увидеть карточки с информацией по каждому пользователю, зарегистрированному в системе приложения. Сверху над ними можно увидеть поле, по которому можно осуществлять поиск по имени, фамилии, отчеству или по всем трем пунктам сразу.

6.3.4 Смена пароля

На главной странице есть кнопка «Смена пароля», для авторизованного пользователя.

Кнопка вызывает всплывающее окно, в которое необходимо ввести новый пароль. Для подтверждения ввести старый пароль еще раз и повторить новый. Эти поля можно увидеть на рисунке 6.10

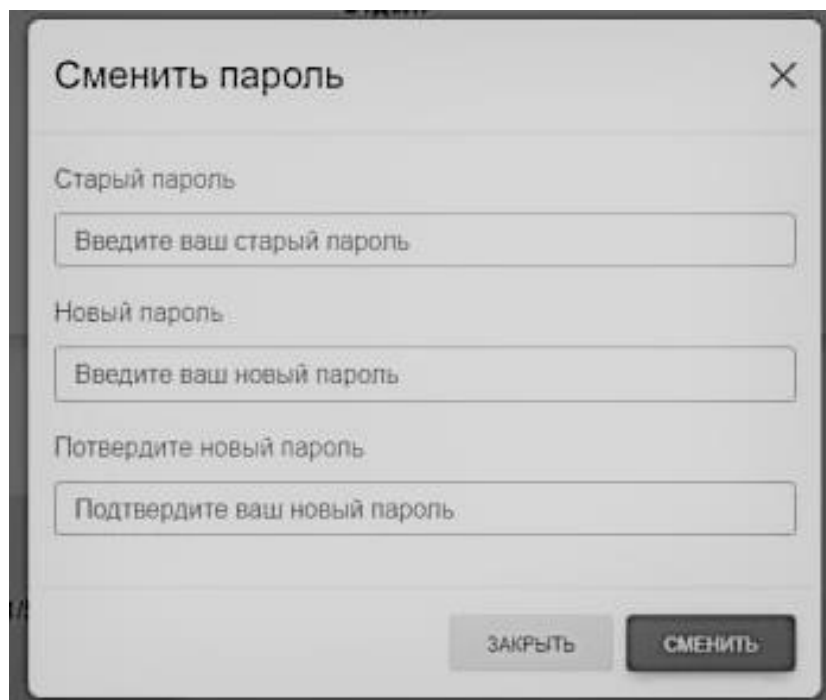


Рисунок 6.10 – Смена пароля

6.3.5 Заблокированные пользователи

Заблокированные пользователи не смогут войти в систему еще на уровне авторизации, у них будет появляться оповещение о том, что их аккаунт был заблокирован и им необходимо обратиться в поддержку.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ЛИЧНОГО КАБИНЕТА СОТРУДНИКА БГУИР

7.1 Характеристика разработанного по индивидуальному заказу веб-приложения

Разрабатываемое в дипломном проекте программное средство представляет собой веб-приложение личного кабинета для сотрудника БГУИР. В ходе разработки программной части получен программный комплекс, реализованный как набор программ, выполняющих функции проекта, позволяющего манипулировать необходимыми данными и способствовать лучшему взаимодействию сотрудников.

Веб-приложение реализуется по индивидуальному заказу университета кампанией-разработчиком. Главными требованиями, положенными в основу при разработке комплекса, стали интуитивно понятный и легкий в использовании интерфейс, расширяемый функционал, а также охват нужд сотрудников и облегчение их коммуникаций.

Задачей данного программного продукта является предоставление всей информации, необходимой для рабочего процесса, вывод оповещений и облегчение процесса документации. Наличие базы данных сотрудников и манипулирование данными определенным пользователям, чтобы данные отдела кадров и базы данных в приложении совпадали и не вызывали конфликтов. Визуальная концепция приложения должна быть комфортной и понятной, ориентированный на широкий охват.

Программное средство позволяет обеспечить более тесное взаимодействие сотрудников друг с другом через Интернет. Сотрудники могут легко и своевременно получать необходимую для рабочего процесса информацию, вроде данных друг о друге, о различных событиях.

Разработанный программный комплекс соответствует требованиям пользователей, не нуждается в большом количестве аппаратных ресурсов, основан на технологии, не зависящей от выбора платформы. Данные характеристики подразумевают под собой то, что приложение не требует большое количество ресурсов и затрат. Облегчение коммуникации, возможность отправлять информацию в личный кабинет, где сотрудник сможет просмотреть в удобное ему время данные без необходимости тратить время на телефонные звонки и личные встречи будет также иметь значительный экономический эффект.

7.2 Расчет затрат на разработку и цены программного средства, созданного по индивидуальному заказу

Для реализации данного проекта компания-разработчик заключила соглашение с компанией-заказчиком на разработку веб-приложения. В

соглашении определены требования к программному средству и установлена цена.

Цена программного средства будет определена на основе полных затрат на разработку организацией-разработчиком ($C_{отп}$).

7.2.1 Затраты на основную заработную плату команды разработчиков

Для расчета затрат на разработку программного средства в первую очередь необходимо рассчитать основную заработную плату команды разработчиков.

Расчет осуществляется по формуле:

$$Z_o = K_{пр} \sum_{i=1}^n Z_{чи} \cdot t_i, \quad (7.1)$$

где $K_{пр}$ – коэффициент премий;

n – категории исполнителей, занятых разработкой программного средства;

$Z_{чи}$ – часовая заработная плата исполнителя i -й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i -й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч.

Расчет часовой заработной платы каждого исполнителя производится путем деления его месячной заработной платы (оклад плюс надбавки) на количество рабочих часов в месяце (расчетная норма рабочего времени для пятидневной недели составляет 168 часов).

Состав команды разработчиков включает в себя обычного и разработчика серверной части и разработчика клиенткой части, а также тестировщика. Размеры заработных плат сотрудников указаны по состоянию на 17.04.2023.

Расчет затрат на основную заработную плату разработчиков представлен в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату разработчиков

Категория исполнителя	Месячный оклад, руб.	Часовой оклад, руб.	Трудоемкость работ, ч	Итого, руб.
Разработчик клиентской части	2050,00	12,20	230	2806,00
Тестировщик	1200,00	7,14	100	714,00
Разработчик серверной части	3010,00	17,92	180	3225,60
Итого				6745,60
Премия (30% от основной заработной платы)				2023,70
Всего затрат на заработную плату разработчиков				8769,30

7.2.2 Затраты на дополнительную заработную плату команды

Дополнительная заработная плата определяется по нормативу в процентах к основной заработной плате по следующей формуле:

$$З_д = \frac{З_о \cdot Н_д}{100\%}, \quad (7.2)$$

где $З_о$ – затраты на основную заработную плату;

$Н_д$ – норматив дополнительной заработной платы, 15%.

Подставим имеющиеся значения в формулу 7.2 и получим расчет:

$$З_д = \frac{8769,30 \cdot 15\%}{100\%} = 1315,40 \text{ руб.}$$

Затраты на дополнительную заработную плату составят 1315,40 руб.

7.2.3 Отчисления на социальные нужды

Расчет отчислений в фонд социальной защиты населения и на обязательное страхование определяется в соответствии с действующими законодательными актами Республики Беларусь и вычисляются по формуле:

$$Р_{\text{соц}} = \frac{(З_о + З_д) \cdot Н_{\text{соц}}}{100\%}, \quad (7.3)$$

где $Н_{\text{соц}}$ – норматив отчислений на социальные нужды, %.

Согласно законодательству Республики Беларусь, отчисления на социальные нужды составляют 34% в фонд социальной защиты и 0,6% на обязательное страхование. Подставим имеющиеся значения в формулу 7.3 и получим результат:

$$Р_{\text{соц}} = \frac{(8769,30 + 1315,40) \cdot 34,6\%}{100\%} = 3489,30 \text{ руб.}$$

Размер отчислений в фонд социальной защиты и на обязательное страхование составит 3489,30 руб.

7.2.4 Прочие расходы

Расходы по данной статье осуществляется в процентах от затрат на основную заработную плату команды разработчиков и рассчитывается по формуле:

$$P_{\text{пр}} = \frac{Z_o \cdot H_{\text{пз}}}{100\%}, \quad (7.4)$$

где $H_{\text{пз}}$ – норматив прочих затрат в целом по организации, 40%;

Подставим имеющиеся значения в формулу 7.4 и произведем расчет.

$$P_{\text{пр}} = \frac{8769,30 \cdot 40\%}{100\%} = 3507,72 \text{ руб.}$$

Размер прочих расходов составит 3507,72 рублей.

7.2.5 Общая сумма затрат на разработку

Общая сумма затрат на разработку рассчитывается путем суммирования всех статей затрат. Представим в виде формулы:

$$Z_p = Z_o + Z_d + P_{\text{соц}} + P_{\text{пр}}. \quad (7.5)$$

Подставим имеющиеся значения в формулу 7.5 и произведем расчет.

$$Z_p = 8769,30 + 1315,40 + 3489,30 + 3507,72 = 17081,72 \text{ руб.}$$

Согласно расчетам, общая сумма затрат на разработку составит 17081,72 рублей.

7.2.6 Плановая прибыль, включаемая в цену программного средства

Плановая прибыль, включаемая в цену программного средства, рассчитывается по формуле:

$$P_{\text{п.с}} = \frac{Z_p \cdot P_{\text{п.с}}}{100\%}, \quad (7.6)$$

где $P_{\text{п.с}}$ – рентабельность затрат на разработку программного средства, 30%

В данном случае рентабельность установили на уровне 30%. Подставим имеющиеся значения в формулу 7.6 и произведем расчет.

$$P_{\text{п.с}} = \frac{17081,72 \cdot 30\%}{100\%} = 5124,52 \text{ руб.}$$

Исходя из расчетов, плановая прибыль, включаемая в цену программного средства, составит 5124,52 рублей.

7.2.7 Отпускная цена программного средства

Отпускная цена программного продукта представляет собой сумму затрат на заработную плату и плановой прибыли. Рассмотрим основную формулу:

$$Ц_{п.с} = З_p + П_{п.с} , \quad (7.7)$$

Подставим имеющиеся значения в формулу 7.7, произведем расчеты.

$$Ц_{п.с} = 17081,72 + 5124,52 = 22216,24 \text{ руб.}$$

Расчет цены на разработку программного средства личного кабинета сотрудника БГУИР представлен в итоговой таблице 7.2.

Таблица 7.2 – Результаты расчета цены на разработку программного средства.

Наименование статьи затрат	Сумма, р.
1. Основная заработная плата разработчиков	8769,30
2. Дополнительная заработная плата разработчиков	1315,40
3. Отчисления на социальные нужды	3489,30
4. Прочие расходы	3507,72
5. Всего затраты на разработку	17081,72
6. Плановая прибыль	5124,52
7. Цена программного средства	22216,24

Таким образом, отпускная цена программного средства личного кабинета сотрудника БГУИР составит 22216,24 рублей, при затратах на разработку в 17081,72 рублей.

Ввиду выполненного объема работы и затраченных человеко-часов данная цена является среднерыночной и удовлетворительной. Данный вывод можно подтвердить, проведя расчет экономической эффективности разработки данного программного средства.

7.3. Расчет результата от разработки и реализации программного модуля личного кабинета сотрудника БГУИР

В данном случае организация выступает в лице разработчика программного средства по индивидуальному заказу. Для организации-разработчика экономическим эффектом является прирост чистой прибыли, полученной от разработки и реализации программного средства заказчику. Так как программное средство будет реализовываться организацией-разработчиком по отпускной цене, сформированной на основе затрат на

разработку, то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле:

$$\Delta\Pi_{\text{ч}} = \Pi_{\text{п.с}} \left(1 - \frac{H_{\text{п}}}{100}\right), \quad (7.8)$$

где $\Pi_{\text{п.с}}$ – прибыль, включаемая в цену программного средства, р;

$H_{\text{п}}$ – ставка налога на прибыль согласно действующему законодательству.

Подставим имеющиеся данные в формулу 7.8 и произведем расчет:

$$\Delta\Pi_{\text{ч}} = 5124,52 \left(1 - \frac{20\%}{100\%}\right) = 4100,42 \text{ руб.}$$

Экономический эффект, полученный в виде прироста чистой прибыли от его разработки и реализации заказчику, составит 4100,42 рублей.

7.4. Расчет показателей экономической эффективности разработки программного модуля личного кабинета сотрудника БГУИР

Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета рентабельности затрат на разработку программного средства. Рентабельность является одним из основных показателей эффективности предприятия с точки зрения использования привлеченных средств. Она представляет собой отношение суммы чистой приведенной прибыли, полученной за весь расчетный период, к суммарным приведенным затратам за этот же период и определяется по формуле:

$$P_3 = \frac{\Delta\Pi_{\text{ч}}}{Z_{\text{р}}} \cdot 100\%, \quad (7.9)$$

где $\Delta\Pi_{\text{ч}}$ – прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, руб.;

$Z_{\text{р}}$ – затраты на разработку программного средства организацией-разработчиком, руб.

Подставим имеющиеся данные в формулу 7.9 и произведем расчет.

$$P_3 = \frac{4100,42}{17081,72} \cdot 100\% = 24\%.$$

Рассчитанный показатель отображает, сколько чистой прибыли компания-разработчик получит от вложенных денег в разработку программного средства и составляет 24%.

В результате проведения расчетов была определена необходимость разработки программного обеспечения, а также получен экономический эффект от использования данного программного продукта. По результатам проведенного экономического обоснования были получены следующие результаты:

1. Затраты на разработку программного средства составят 17081,72 рублей.

2. Веб-приложение личного кабинета сотрудника БГУИР будет реализовано заказчику по цене 22216,24 рублей.

3. Прирост чистой прибыли от его разработки и реализации заказчику составит 4100,42 рубля.

4. Данная разработка имеет положительный экономический эффект в виде чистой прибыли, которую компания-разработчик получит от вложенных денег в разработку программного средства, в размере 24%.

Таким образом, разработка и реализация по индивидуальному заказу программного модуля личного кабинета сотрудника БГУИР с экономической точки зрения целесообразна.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта было разработано веб-приложение, реализующее функционал личного кабинета сотрудника БГУИР. Для его реализации проведена работа по разработке серверной и клиентской частей.

Для написания проекта использовался язык программирования Java, библиотека Spring для серверной части и JavaScript, библиотека React для разработки пользовательского интерфейса. В качестве базы данных была выбрана PostgreSQL. Данный набор технологий обоснован простотой и использованием внутри систем БГУИР, что способствует удобной интеграции с уже существующими системами.

В процессе разработки были достигнуты следующие цели:

1. Реализована серверная часть приложения на Java Spring, включающая в себя модули аутентификации и авторизации с использованием JSON Web Token (JWT), а также обработку запросов от клиентской части приложения.
2. Спроектирована и настроена базы данных PostgreSQL.
3. Разработана клиентская часть приложения.
4. Установлена связь между клиентской и серверной частями.
5. Произведено тестирование приложения.

Результатом работы является веб-приложение личного кабинета сотрудника БГУИР, которое позволяет пользователям авторизоваться, просматривать и редактировать свои личные данные, а также выполнять другие задачи, связанные с их профилем и профилями других сотрудников.

После проведения анализа экономической эффективности можно сделать вывод, что проектирование и разработка данного приложения являются выгодными и целесообразными. Оно принесет пользу как разработчику, так и потенциальным пользователям приложения.

Дипломный проект успешно завершен, и все поставленные задачи были выполнены с использованием программных решений. Разработано приложение с гибкой и расширяемой архитектурой, которая обеспечивает возможность дальнейшего расширения и дополнения функциональности при необходимости. Проект готов к дальнейшему улучшению и развитию путем внедрения дополнительных компонентов и поддержки альтернативных методов ввода и обработки данных.

Заключительный плакат представлен на чертеже ГУИР.400201.307 ПЛ.2.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Sage HR [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://sage.hr/ru>. – Дата доступа: 01.04.2023.
- [2] WebHR [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://web.hr>. – Дата доступа: 01.04.2023.
- [3] Архитектура веб-приложений: принципы, протоколы, практика. / Л. Шкляр Р. Розен – Эксмо, 2011. – 634 с.
- [4] Клиент-серверная архитектура [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit. – Дата доступа: 01.04.23.
- [5] Трехуровневая клиент-серверная архитектура [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://testmatick.com/ru/osnovnye-ponyatiya-i-osobennosti-klient-servernoj-arhitektury/>. – Дата доступа: 01.04.23.
- [6] Паттерны объектно-ориентированного проектирования. / Э. Гамма Р. Хелм Р. Джонсон Дж. Влиссидес – СПб. Издательство Питер, 2020. – 448с.
- [7] Принцип SOLID [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://web-creator.ru/articles/mvc>. – Дата доступа: 01.04.23.
- [8] PostgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/>. – Дата доступа: 01.04.2023.
- [9] Spring Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://spring.io/why-spring>. – Дата доступа: 01.04.23.
- [10] React [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.reactjs.org>. – Дата доступа: 01.04.23.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

com/epa/epadiplom/config/ApplicationConfig.java

```
package com.epa.epadiplom.config;

import com.epa.epadiplom.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {
    private final UserService userService;

    @Bean
    public UserDetailsService userDetailsService() {
        return userService::findUserByFirstName;
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
        DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager
    authenticationManager(AuthenticationConfiguration config)
        throws Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

com/epa/epadiplom/config/JwtAuthenticationFilter.java

```
package com.epa.epadiplom.config;

import com.epa.epadiplom.service.JwtService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.lang.NonNull;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
    @NonNull HttpServletResponse response, @NonNull FilterChain filterChain)
    throws ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwtToken;
        final String login;
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwtToken = authHeader.substring(7);
        login = jwtService.extractUsername(jwtToken);
        if (login != null &&
            SecurityContextHolder.getContext().getAuthentication() ==
            null) {
            UserDetails userDetails =
            this.userDetailsService.loadUserByUsername(login);
            if (jwtService.isTokenValid(jwtToken, userDetails)) {
                UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities());
                authenticationToken.setDetails(
```

```

        new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authenticationToken);
    }
    }
    filterChain.doFilter(request, response);
}
}

```

com/epa/epadiplom/config/SecurityConfig.java

```

package com.epa.epadiplom.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;
import org.springframework.security.web.session.SessionManagementFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import java.util.Arrays;
import java.util.List;

@Configuration
@EnableWebSecurity(debug = true)
@RequiredArgsConstructor
public class SecurityConfig implements WebMvcConfigurer {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final AuthenticationProvider authenticationProvider;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity)
throws Exception {
        httpSecurity
            .cors(Customizer.withDefaults())
            .and()
            .addFilterBefore(new CorsFilter(corsConfigurationSource()),
SessionManagementFilter.class)
            .httpBasic().disable()
            .csrf().disable()

```

```

        .authorizeHttpRequests()
        .requestMatchers("/**")
        .permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .authenticationProvider(authenticationProvider)
        .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
    return httpSecurity.build();
}

@Bean
CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(List.of("http://localhost:3000"));
    configuration.setAllowedMethods(Arrays.asList("GET", "POST"));
    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
}

```

com/epa/epadiplom/controller/AuthenticationController.java

```

package com.epa.epadiplom.controller;

import com.epa.epadiplom.entity.authentication.AuthenticationRequest;
import com.epa.epadiplom.entity.authentication.AuthenticationResponse;
import com.epa.epadiplom.entity.authentication.RegisterRequest;
import com.epa.epadiplom.service.AuthenticationService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthenticationController {
    private final AuthenticationService authenticationService;

    @PostMapping("/register")
    public ResponseEntity<AuthenticationResponse> register (
        @RequestBody RegisterRequest request){
        return ResponseEntity.ok(authenticationService.register(request));
    }

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> authenticate (
        @RequestBody AuthenticationRequest request){
        return
ResponseEntity.ok(authenticationService.authenticate(request));
    }
}

```

```
}
```

com/epa/epadiplom/controller/MainPageController.java

```
package com.epa.epadiplom.controller;

import com.epa.epadiplom.entity.*;
import com.epa.epadiplom.entity.authentication.*;
import com.epa.epadiplom.service.*;
import lombok.RequiredArgsConstructor;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.core.Authentication;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import java.util.List;

@RestController
@EnableMethodSecurity
@EnableWebMvc
@RequiredArgsConstructor
public class MainPageController {
    private final EmployeesViewService employeesViewService;
    private final EmployeeFullViewService employeeFullViewService;
    private final LogStatementsViewService logStatementsViewService;
    private final EventsViewService eventsViewService;
    private final TasksViewService tasksViewService;
    private final UserService userService;
    private final LogStatementService logStatementService;
    private final EventService eventService;
    private final NoticeEventService noticeEventService;
    private final TaskService taskService;
    private final EmployeeTaskService employeeTaskService;
    private final PasswordEncoder passwordEncoder;
    private final DocumentService documentService;
    private final EmployeeService employeeService;
    private final PersonalService personalService;
    // _____
    // Request to see the main info of authorized employee page
    @GetMapping(path = "/mainUserInfo", produces =
    MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<EmployeeFullView>>
    getEmployeeInfo(Authentication authentication) {
        return
    ResponseEntity.ok(employeeFullViewService.findAllByLoginUser(authentication.getName()));
    }

    // _____
    // Request to see the statements that need to approve
    @GetMapping(path = "/ls", produces = MediaType.APPLICATION_JSON_VALUE)
```



```

        public ResponseEntity<List<LogStatementsView>>
getLsRequests(Authentication authentication) {
            return
ResponseEntity.ok(this.logStatementsViewService.findAllByIdApproverAndStatus(
userService.findUserByFirstName(authentication.getName()).getIdLogin(), 3));
        }

//      Request to approve the statements
        @PostMapping(path = "/ls/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> setLsApprove(
            @PathVariable Long id,
            Authentication authentication,
            @RequestBody LogStatementRequest request
        ) {
            LogStatement logStatement =
logStatementService.findByIdAndIdApprover(
                id,

userService.findUserByFirstName(authentication.getName()).getIdLogin()
                ).orElse(new LogStatement());
            logStatement.setStatus(request.getStatus());
            System.out.println(logStatement.getId() + "    " +
                logStatement.getStatus());
            if(logStatementService.saveLogStatement(logStatement))
                System.out.println("ok");
            return ResponseEntity.ok("Done ");
        }

// Request to see the events for authorized employee
        @GetMapping(path = "/events", produces =
MediaType.APPLICATION_JSON_VALUE)
        public @ResponseBody List<EventsView> getEvents(Authentication
authentication) {
            return this.eventsViewService.findAllByIdRecipient(

userService.findUserByFirstName(authentication.getName()).getIdLogin());
        }

// Request to see the tasks for authorized employee
        @GetMapping(path = "/tasks", produces = MediaType.APPLICATION_JSON_VALUE)
        public @ResponseBody List<TasksView> getTasks(Authentication
authentication) {
            System.out.println(authentication);
            return this.tasksViewService.findAllByIdExecutor(

userService.findUserByFirstName(authentication.getName()).getIdLogin());
        }

// _____
// Request for seeing list of all employees
        @GetMapping(path = "/employees", produces =
MediaType.APPLICATION_JSON_VALUE)
        public @ResponseBody List<EmployeesView> getEmployees() {
            return this.employeesViewService.findAll();
        }

```

```

        @PostMapping(path = "/event", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> createEvent(@RequestBody EventRequest
request) {
            Event event = Event.builder()
                .dateOfEvent(request.getDateOfEvent())
                .commentFe(request.getCommentFe())
                .typeOfEvent(request.getTypeOfEvent())
                .build();
            if(eventService.saveEvent(event))
                System.out.println("ok");
            return ResponseEntity.ok("Done");
        }

        @PostMapping(path = "/noticeevent/{idevent}", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> createNoticeEvent(
            @RequestBody NoticeEventRequest request,
            Authentication authentication,
            @PathVariable Long idevent
        ) {
            NoticeEvent noticeEvent = NoticeEvent.builder()
                .recipientId(request.getRecipientId())
                .eventId(idevent)

.employeeId(userService.findUserByFirstName(authentication.getName()).getIdLo
gin())
                .build();
            System.out.println(noticeEvent.getRecipientId() + " " +
                noticeEvent.getEventId() + " " +
                noticeEvent.getEmployeeId());
            if(noticeEventService.saveNoticeEvent(noticeEvent))
                System.out.println("ok");
            return ResponseEntity.ok("Done");
        }

        @PostMapping(path = "/task", produces = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> createTask(@RequestBody TaskRequest
request) {
            Task task = Task.builder()
                .dateTask(request.getDateTask())
                .nameOfTask(request.getNameOfTask())
                .idExecutor(request.getIdExecutor())
                .commentTe(request.getCommentTe())
                .build();
            System.out.println(task.getId());
            EmployeeTask employeeTask = EmployeeTask.builder()
                .idTask(task.getId())
                .idEmployee(request.getIdExecutor())
                .build();
            if (taskService.saveTask(task))
                System.out.println("ok");
            if (employeeTaskService.saveEmployeeTask(employeeTask))
                System.out.println("ok");
            return ResponseEntity.ok("Done");
        }
    }

```

```

        @PostMapping(path = "/password", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> changePassword(@RequestBody LoginRequest
request,
                                                    Authentication authentication)
        {
            User user =
userService.findUserByFirstName(authentication.getName());
            user.setPassword(passwordEncoder.encode(request.getPassword()));
            if (userService.saveUserPassword(user))
                System.out.println("ok");
            return ResponseEntity.ok("Done");
        }

        @PostMapping(path = "/lscreate", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> createLS( @RequestBody
LogStatementCreateRequest request,
                                                    Authentication authentication
                                                    ){
            LogStatement logStatement = LogStatement.builder()
                .id(0)
                .idApprover(request.getIdApprover())
                .status(3)

                .idEmployee(userService.findUserByFirstName(authentication.getName()).getIdLo
gin())

                .commentLs(request.getCommentLs())
                .typeLeave(request.getTypeLeave())
                .dateLeave(request.getDateLeave())
                .dateOfLs(request.getDateOfLs())
                .daysSum(request.getDaysSum())
                .build();
            if(logStatementService.saveLogStatementAll(logStatement))
                System.out.println("ok");
            if(request.getBodyDoc() != null) {
                Document document = Document.builder()
                    .bodyDoc(request.getBodyDoc())
                    .idLs(logStatement.getId())
                    .build();
                if(documentService.saveDocument(document))
                    System.out.println("ok");
            }
            return ResponseEntity.ok("Done ");
        }

        @PostMapping(path = "/editemployee/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<String> edit( @RequestBody EditRequest request,
                                                    Authentication authentication,
                                                    @PathVariable Long id
                                                    ){
            Employee employee = Employee.builder()
                .id(id)
                .firstName(request.getFirstName())
                .lastName(request.getLastName())
                .middleName(request.getMiddleName())

```

```

        .workNumber(request.getWorkNumber())
        .locationStreet(request.getLocationStreet())
        .cabinetOffice(request.getCabinetOffice())
        .build();
    if (employeeService.saveEmployee(Employee))
        System.out.println("ok");
    Personal personal = Personal.builder()
        .birthD(request.getBirthD())
        .idPersonal(id)
        .personalNumber(request.getPersonalNumber())
        .build();
    if (personalService.savePersonal(Personal))
        System.out.println("ok");
    User user = User.builder()
        .idLogin(id)
        .role(request.getRole())
        .build();
    if (userService.saveUser(User))
        System.out.println("ok");
    return ResponseEntity.ok("Done ");
}
}

```

com/epa/epadiplom/entity/authentication/AuthenticationRequest.java

```

package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AuthenticationRequest {
    private String login;
    String password;
}

```

com/epa/epadiplom/entity/authentication/AuthenticationResponse.java

```

package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor

```

```
public class AuthenticationResponse {
    private String token;
}
```

com/epa/epadiplom/entity/authentication/EditRequest.java

```
package com.epa.epadiplom.entity.authentication;

import com.epa.epadiplom.entity.employeeAttributes.Role;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import java.sql.Date;

@Data
@Builder
@NoArgsConstructor
@NoArgsConstructor
public class EditRequest {

    private String firstName;
    private String middleName;
    private String lastName;
    private Date birthD;
    private Role role;
    private long workNumber;
    private long personalNumber;
    private String locationStreet;
    private String cabinetOffice;

}
```

com/epa/epadiplom/entity/authentication/EventRequest.java

```
package com.epa.epadiplom.entity.authentication;

import lombok.*;

import java.sql.Date;

@Data
@Builder
@NoArgsConstructor
@NoArgsConstructor
public class EventRequest {

    private String typeOfEvent;
    private String commentFe;
    private Date dateOfEvent;

}
```

com/epa/epadiplom/entity/authentication/LoginRequest.java

```
package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class LoginRequest {

    private String password;
}
```

com/epa/epadiplom/entity/authentication/LogStatementCreateRequest.java

```
package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.sql.Date;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class LogStatementCreateRequest {

    private long idApprover;
    private String commentLs;
    private int daysSum;
    private int typeLeave;
    private Date dateLeave;
    private Date dateOfLs;
    private String bodyDoc;
}
```

com/epa/epadiplom/entity/authentication/LogStatementRequest.java

```
package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class LogStatementRequest {

    private int status;

}

```

com/epa/epadiplom/entity/authentication/NoticeEventRequest.java

```

package com.epa.epadiplom.entity.authentication;

import lombok.*;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class NoticeEventRequest {

    private long recipientId;

}

```

com/epa/epadiplom/entity/authentication/RegisterRequest.java

```

package com.epa.epadiplom.entity.authentication;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class RegisterRequest {

    private String firstName;
    private String middleName;
    private String lastName;
    private String login;
    private String mail;
    private String password;

}

```

com/epa/epadiplom/entity/authentication/TaskRequest.java

```

package com.epa.epadiplom.entity.authentication;

```

```

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.sql.Date;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TaskRequest {

    private Date dateTask;
    private String nameOfTask;
    private String commentTe;
    private long idExecutor;

}

```

com/epa/epadiplom/entity/User.java

```

package com.epa.epadiplom.entity;

import com.epa.epadiplom.entity.employeeAttributes.Role;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import jakarta.persistence.*;
import lombok.*;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.List;

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
//To suppress serializing properties with null values
@JsonSerialize
//Ignoring new fields on JSON objects
@JsonIgnoreProperties(ignoreUnknown = true)
@Entity
@Table(name = "login", schema = "public", catalog = "EPA")
public class User implements UserDetails {
    @Id
    //@GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_login")
    private long idLogin;
    @Column(name = "login_user")
    private String firstName;
    @Column(name = "password_user")
    private String password;
}

```



```

@Column(name = "mail_user")
private String mail;
@Enumerated(EnumType.STRING)
private Role role;

@OneToOne
@JoinColumn(name = "id_login")
private Employee employee;

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return List.of(new SimpleGrantedAuthority(role.name()));
}

@Override
public String getUsername() {
    return firstName;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
}

```

com/epa/epadiplom/service/AuthenticationService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Employee;
import com.epa.epadiplom.entity.User;
import com.epa.epadiplom.entity.authentication.AuthenticationRequest;
import com.epa.epadiplom.entity.authentication.AuthenticationResponse;
import com.epa.epadiplom.entity.authentication.RegisterRequest;
import com.epa.epadiplom.entity.employeeAttributes.Role;
import com.epa.epadiplom.repository.EmployeeRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

```

```

import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class AuthenticationService {
    private final UserService userService;
    private final EmployeeService employeeService;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;
    private final EmployeeRepository employeeRepository;

    public AuthenticationResponse register(RegisterRequest request) {
        Employee employee = employeeRepository
            .findByFirstNameAndMiddleNameAndLastName(
                request.getFirstName(),
                request.getMiddleName(),
                request.getLastName()).orElse(new Employee());
        var user = User.builder()
            .idLogin(employee.getId())
            .firstName(request.getLogin())
            .mail(request.getMail())
            .password(passwordEncoder.encode(request.getPassword()))
            .role(Role.USER)
            .build();
        userService.saveUser(user);
        var jwtToken = jwtService.generateToken(user);
        return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
    }

    public AuthenticationResponse authenticate(AuthenticationRequest request)
    {
        authenticationManager.authenticate(new
        UsernamePasswordAuthenticationToken(request.getLogin(),
        request.getPassword()));
        var user = userService.findUserByFirstName(request.getLogin());
        var jwtToken = jwtService.generateToken(user);
        return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
    }
}

```

com/epa/epadiplom/service/DepartmentService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.repository.DepartmentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor

```

```

public class DepartmentService {

    private final DepartmentRepository departmentRepository;

}

```

com/epa/epadiplom/service/DocumentService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Document;
import com.epa.epadiplom.repository.DocumentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class DocumentService {

    private final DocumentRepository documentRepository;

    public boolean saveDocument(Document document){
        if(documentRepository.findById(document.getId()).isPresent()) {

            return false;
        }
        documentRepository.save(document);
        return true;
    }
}

```

com/epa/epadiplom/service/EmployeeFullViewService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.EmployeeFullView;
import com.epa.epadiplom.repository.EmployeeFullViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class EmployeeFullViewService {

    private final EmployeeFullViewRepository employeeFullViewRepository;

    public List<EmployeeFullView> findAllByLoginUser(String loginUser){
        return employeeFullViewRepository.findAllByLoginUser(loginUser);
    }

}

```

com/epa/epadiplom/service/EmployeeService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Employee;
import com.epa.epadiplom.repository.EmployeeRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class EmployeeService {

    private final EmployeeRepository employeeRepository;

    public boolean saveEmployee(Employee employee) {
        if (employeeRepository.findById(employee.getId()).isPresent()) {
            employeeRepository.save(employee);
            return true;
        }
        return false;
    }
}
```

com/epa/epadiplom/service/EmployeesViewService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.EmployeesView;
import com.epa.epadiplom.repository.EmployeesViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class EmployeesViewService {
    private final EmployeesViewRepository employeesViewRepository;
    public List <EmployeesView> findAll(){
        return employeesViewRepository.findAll();
    }
}
```

com/epa/epadiplom/service/EmployeeTaskService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.EmployeeTask;
import com.epa.epadiplom.repository.EmployeeTaskRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
```

```

@Service
@RequiredArgsConstructor
public class EmployeeTaskService {

    private final EmployeeTaskRepository employeeTaskRepository;

    public boolean saveEmployeeTask (EmployeeTask employeeTask) {
        if (employeeTaskRepository.findById(employeeTask.getId()).isPresent())
            return false;
        employeeTaskRepository.save(employeeTask);
        return true;
    }
}

```

com/epa/epadiplom/service/EventService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Event;
import com.epa.epadiplom.repository.EventRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class EventService {

    private final EventRepository eventRepository;

    public boolean saveEvent (Event event) {
        if (eventRepository.findById(event.getId()).isPresent())
            return false;
        eventRepository.save(event);
        return true;
    }
}

```

com/epa/epadiplom/service/EventsViewService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.EventsView;
import com.epa.epadiplom.repository.EventsViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class EventsViewService {
    private final EventsViewRepository eventsViewRepository;
}

```

```

        public List<EventsView> findAllByIdRecipient(long idLogin) {
            return eventsViewRepository.findAllByIdRecipient(idLogin);
        }
    }
}

```

com/epa/epadiplom/service/JobEmployeeService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.repository.JobEmployeeRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class JobEmployeeService {

    private final JobEmployeeRepository jobEmployeeRepository;
}

```

com/epa/epadiplom/service/JobTitleService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.repository.JobTitleRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class JobTitleService {

    private final JobTitleRepository jobTitleRepository;
}

```

com/epa/epadiplom/service/JobTitleViewService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.repository.JobTitleViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class JobTitleViewService {

    private final JobTitleViewRepository jobTitleViewRepository;
}

```

com/epa/epadiplom/service/JwtService.java

```
package com.epa.epadiplom.service;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Service
public class JwtService {
    private static final String SECRET_KEY =
"77217A25432A462D4A614E645267556B58703273357538782F413F4428472B4B";

    public String extractUsername(String jwtToken) {
        return extractClaim(jwtToken, Claims::getSubject);
    }

    public <T> T extractClaim(String jwtToken, Function<Claims, T>
claimsResolver){
        final Claims claims = extractAllClaims(jwtToken);
        return claimsResolver.apply(claims);
    }

    public String generateToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails
    ){
        return Jwts
            .builder()
            .setClaims(extraClaims)
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 1200 *
60 * 24))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
    }

    public String generateToken(UserDetails userDetails){
        return generateToken(new HashMap<>(), userDetails);
    }

    public boolean isTokenValid (String jwtToken, UserDetails userDetails){
        final String username = extractUsername(jwtToken);
        return (username.equals(userDetails.getUsername())) &&
            !isTokenExpired(jwtToken);
    }
}
```

```

private boolean isTokenExpired(String jwtToken) {
    return extractExpiration(jwtToken).before(new Date());
}

private Date extractExpiration(String jwtToken) {
    return extractClaim(jwtToken, Claims::getExpiration);
}

private Claims extractAllClaims(String jwtToken) {
    return Jwts
        .parserBuilder()
        .setSigningKey(getSignInKey())
        .build()
        .parseClaimsJws(jwtToken)
        .getBody();
}

private Key getSignInKey() {
    byte[] keyBytes = Decoders.BASE64.decode(SECRET_KEY);
    return Keys.hmacShaKeyFor(keyBytes);
}
}

```

com/epa/epadiplom/service/LogStatementService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.LogStatement;
import com.epa.epadiplom.repository.LogStatementRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@RequiredArgsConstructor
public class LogStatementService {

    private final LogStatementRepository logStatementRepository;

    public boolean saveLogStatement(LogStatement logStatement) {
        if(logStatementRepository.findById(logStatement.getId()).isPresent())
        {
            logStatementRepository.save(logStatement);
            return true;
        }
        return false;
    }

    public boolean saveLogStatementAll(LogStatement logStatement){
        if(logStatementRepository.findById(logStatement.getId()).isPresent())
        {
            return false;
        }
        logStatementRepository.save(logStatement);
        return true;
    }
}

```



```

    }

    public Optional<LogStatement> findByIdAndIdApprover (long id, long
idApprover){
        return logStatementRepository.findByIdAndIdApprover(id, idApprover);
    }
}

```

com/epa/epadiplom/service/LogStatementsViewService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.LogStatementsView;
import com.epa.epadiplom.repository.LogStatementsViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class LogStatementsViewService {
    private final LogStatementsViewRepository logStatementsViewRepository;

    public List<LogStatementsView> findAllByIdApproverAndStatus(long
idApprover, int status){
        return
logStatementsViewRepository.findAllByIdApproverAndStatus(idApprover, status);
    }
}

```

com/epa/epadiplom/service/NoticeEventService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.NoticeEvent;
import com.epa.epadiplom.repository.NoticeEventRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class NoticeEventService {

    private final NoticeEventRepository noticeEventRepository;

    public boolean saveNoticeEvent (NoticeEvent noticeEvent){
        if(noticeEventRepository.findById(noticeEvent.getId()).isPresent())
            return false;
        noticeEventRepository.save(noticeEvent);
        return true;
    }
}

```

com/epa/epadiplom/service/PersonalService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Personal;
import com.epa.epadiplom.repository.PersonalRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class PersonalService {

    private final PersonalRepository personalRepository;

    public boolean savePersonal(Personal personal) {
        if (personalRepository.findById(personal.getIdPersonal())) {
            personalRepository.save(personal);
            return true;
        }
        return false;
    }
}
```

com/epa/epadiplom/service/TaskService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.Event;
import com.epa.epadiplom.entity.Task;
import com.epa.epadiplom.repository.TaskRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class TaskService {

    private final TaskRepository taskRepository;

    public boolean saveTask (Task task){
        if (taskRepository.findById(task.getId()).isPresent())
            return false;
        taskRepository.save(task);
        return true;
    }
}
```

com/epa/epadiplom/service/TasksViewService.java

```
package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.TasksView;
```

```

import com.epa.epadiplom.repository.TasksViewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class TasksViewService {
    private final TasksViewRepository tasksViewRepository;
    public List<TasksView> findAllByIdExecutor(long idLogin) {
        return tasksViewRepository.findAllByIdExecutor(idLogin);
    }
}

```

com/epa/epadiplom/service/UserService.java

```

package com.epa.epadiplom.service;

import com.epa.epadiplom.entity.User;
import com.epa.epadiplom.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;

    public User findUserByFirstName(String login) {
        return userRepository.findByFirstName(login).orElseThrow();
    }

    public List<User> allUsers() {
        return userRepository.findAll();
    }

    public boolean saveUser(User user) {
        if (userRepository.findByFirstName(user.getFirstName()).isPresent())
            return false;
        userRepository.save(user);
        return true;
    }

    public boolean saveUserPassword(User user) {
        if (userRepository.findByFirstName(user.getFirstName()).isPresent()) {
            userRepository.save(user);
            return true;
        }
        return false;
    }
}

```

ПРИЛОЖЕНИЕ Б
(обязательное)

Спецификация программного дипломного проекта

ПРИЛОЖЕНИЕ В
(обязательное)

Ведомость документов

[illegible]

Отчет о проверке на заимствования №1



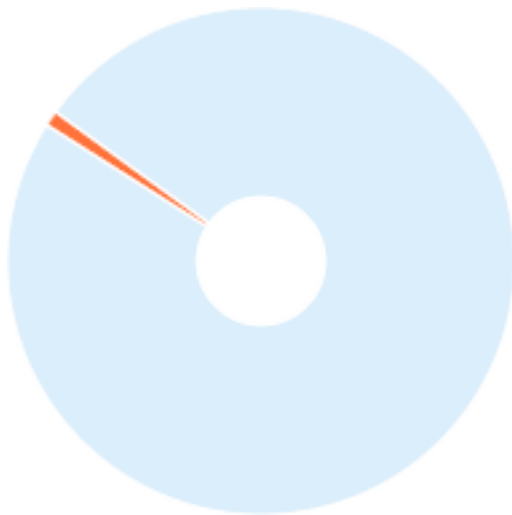
Автор: mint.dragon@mail.ru / ID: 10376112
Проверяющий:
Отчет предоставлен сервисом «Антиплагиат» - <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 12
Начало загрузки: 16.05.2023 22:29:08
Длительность загрузки: 00:00:03
Имя исходного файла:
890541_Игнатович_пояснительная_записка.pdf
Название документа:
890541_Игнатович_Пояснительная записка_Личный кабинет сотрудника БГУИР
Размер текста: 136 кБ
Символов в тексте: 139077
Слов в тексте: 16765
Число предложений: 1055

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 16.05.2023 19:29:12
Длительность проверки: 00:00:05
Комментарии: не указано
Модули поиска: Интернет Free



СОВПАДЕНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
0,49% <div></div>	0% <div></div>	0% <div></div>	99,51% <div></div>

Совпадения - фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» – это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.

Самоцитирования - фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» – это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.

Цитирования - фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» – это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.

Текстовое пересечение - фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.

Источник - документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.

Оригинальный текст - фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» – это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует полному тексту проверяемого документа.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в тексте	Источник	Актуален на	Модуль поиска	Комментарии
[01]	0,49%	Технологии программирования. (Задание 1) - online presentation http://en.ppt-online.org	08 Июл 2018	Интернет Free	
[02]	0,41%	Разработка через тестирование http://ru.wikipedia.org	20 Авг 2020	Интернет Free	Источник исключен. Причина: Маленький процент пересечения.
[03]	0,2%	2_86_109_0_0.600_81676221 Назначения программы «Управления заявками на закупку материальных ценностей» : - регистрация и ввод заявок на закупку тмц отделов (цехов) http://vbibl.ru	06 Дек 2020	Интернет Free	Источник исключен. Причина: Маленький процент пересечения.