МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

МИКРОСЕРВИС ДЛЯ АВТОМАТИЗАЦИИ ИНФОРМИРОВАНИЯ СОТРУДНИКОВ ОРГАНИЗАЦИЙ

Студент 31.05.2022 Бахтияров Р.Р.
(подпись, дата)

Руководитель 31.05.2022 Карамышева Н.С.
(подпись, дата)

Нормоконтролёр 31.05.2022 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 01.06.2022 г. № 13)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 07.06.2022г. № 4)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«26» апреля 20 2 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Бахтияров Руслан Рафикович гр. 18вв2 факультета вычислительной техники.

2.Тема работы «Микросервис для автоматизации информирования сотрудников организации».

Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Карамышева Н.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – микросервис, предназначенный для оповещения сотрудников организаций по электронной почте о назначенных мероприятиях. Приложение должно обладать графическим и интуитивно понятным интерфейсом и *API*, а также безопасной аутентификация и авторизацией.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области: изучить аналоги микросервисов со схожей функциональностью, выделить достоинства и недостатки. Определить архитектуру приложения и средства разработки. Выбрать язык программирования и доступные библиотеки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2022 – 28.05.2022	
2	Анализ требований	32	29.04.2022 – 30.05.2022	
3	Создание макета графического интерфейса	30	1.04.2022 – 3.05.2022	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2022 – 8.05.2022	
5	Реализация графического интерфейса	60	9.05.2022 – 15.05.2022	
6	Разработка основной части программы	76	16.05.2022 – 23.05.2022	
7	Оформление пояснительной записки	54	24.05.2022 – 31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы _____ 26.04.2022 Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент _____ 26.04.2022 Бахтияров Р.Р.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 26.04.2022 Фионова Л.Р.
(подпись, дата

Реферат

Работа содержит 90 листов, 35 рисунков, 5 таблиц, 3 приложения.

МИКРОСЕРВИС, БАЗА ДАННЫХ, СЕРВИС, КОНТРОЛЛЕР, *REST API, JWT, JSON, UI, MySQL, MAVEN, JAVA, UML, CRUD.*

В рамках данной выпускной квалификационной работы был создан микросервис для автоматизации информирования сотрудников организаций.

Цель работы – создать микросервис, отправляющий электронные сообщения сотрудникам организации с удобным пользовательским интерфейсом.

Объектом разработки является микросервис для автоматизации информирования сотрудников организаций.

В результате проведенной работы реализован проект, удовлетворяющий поставленной задаче, а именно: удобный пользовательский интерфейс, автоматизированная отправка сообщений сотрудникам с возможностью сверять сегодняшний день с официальным производственным календарем, безопасная авторизация и аутентификация, открытый *API*.

Разработка проводилась в среде программирования *IntelliJ IDEA Ultimate Edition*. Использованные языки программирования: *Java*.

Созданный микросервис может быть запущен как отдельное приложение, так и внедрен в микросервисную архитектуру на предприятиях, где есть необходимость оповещения большого числа сотрудников.

					ПГУ 1.090301.04.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Бахтияров Р.Р.			Микросервис для автоматизации информирования сотрудников	Лит.	Лист 2 / Листов 123
Проверил		Карамышева Н.С.					
Н. Контр.		Бычков А.С.				ФВТ, каф. ВТ Гр. 18ВВ2	
Утверд.		Митрохин М.А.					

Содержание

Реферат 2

Перечень сокращений и обозначений 5

Введение 6

1 Микросервис для автоматизации информирования сотрудников организации 8

1.1 Анализ предметной области 8

1.2 Постановка задачи 11

1.3 Архитектура приложения 12

1.3.1 Микросервисная архитектура. 12

1.3.2 Шаблон *Controller-Service-Repository* 14

1.4 Инструменты разработки 15

1.4.1 Язык *Java* 15

1.4.2 *Spring Boot* 15

1.4.3 *Spring JDBC* 16

1.4.4 *MySQL* 16

1.4.5 *HikariCP* 17

1.4.6 *REST API* 18

1.4.7 *Spring Security* 19

1.4.8 Стандарт *JWT* 19

1.4.9 Инструмент *Maven* 20

1.5 Реализация 22

1.5.1 Слой хранения данных 22

1.5.2 Слой сервисов 26

1.5.3 Слой контроллеров 28

1.5.4 Безопасность 30
1.6 Описание работы микросервиса 31
2 Охрана труда 48
2.1 Анализ негативных факторов, воздействующие на человека 48
2.1.1 Эргономика рабочего места 48
2.1.2 Правила перерывов и режима работы 49
2.1.3 Электробезопасность 49
2.1.4 Освещенность рабочего места 50
Заключение 52
Список использованных источников 53
Приложение А – UML - диаграммы 54
Приложение Б – Листинг программы 60
Приложение В – Презентация 76

Перечень сокращений и обозначений

JDBC - платформенно независимый промышленный стандарт взаимодействия *Java-приложений* с различными СУБД

UML (*Unified Modeling Language*) – унифицированный язык моделирования

JSON (*JavaScript Object Notation*) – текстовый формат обмена данными, основанный на *JavaScript*

Фреймворк – программная платформа, которая упрощает разработку программного продукта, определяет структуру проекта

SQL (*Structured Query Language*) - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных

JWT (*JSON Web Token*) - это открытый стандарт (*RFC* 7519) для создания секретных ключей доступа, основанный на формате *JSON*

MAVEN - фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM

API (Application Programming Interface) - интерфейс прикладного программирования, который представляет собой набор определений и протоколов для создания и интеграции прикладного программного обеспечения.

REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети

UI (*User Interface*) – пользовательский интерфейс

MySQL - свободная реляционная система управления базами данных

JAR (*Java ARchive*) - это Java-архив

CRUD (*Create*, *Read*, *Update*, *Delete*) – обозначение 4-х основных функций сущности, хранящийся в базе данных

Введение

Бумажные письма в наше время редкость. На смену им пришли электронные сообщения. Электронная почта стала такой огромной частью нашей жизни. Мы считаем само собой разумеющимся, что можем общаться с людьми онлайн 24/7.

Электронная почта начала работать за несколько лет до 1991 года; в тот год она получила публичное представление миру. Электронная почта стала побочным продуктом Всемирной паутины после того, как потребители прониклись многими ее коммуникационными преимуществами. Первыми пользователями платформы обмена сообщениями стали образовательные и военные.

Новаторская инновация получила известность среди деловых кругов по всему миру благодаря своей эффективности в подтверждении поставок, транзакций, маркетинга и другой информации о компании. Электронные письма требовали чрезвычайно низкой стоимости работы, были удобны в использовании и обеспечивали приличную скорость передачи информации. Общение по электронной почте происходит почти мгновенно, что улучшает коммуникацию за счет быстрого распространения информации и быстрого реагирования на запросы клиентов. Это также позволяет быстрее решать проблемы и оптимизировать бизнес-процессы.

Электронная почта снижает географические барьеры и барьеры часовых поясов для бизнеса. Сотрудники или подрядчики со всего мира могут общаться, независимо от их местоположения. Аналогичным образом, клиенты могут отправлять вопросы о продажах и запросы в службу поддержки по электронной почте в соответствии со своими расписаниями часовых поясов.

Электронная почта может помочь повысить производительность. Владельцы бизнеса могут взаимодействовать с установленными списками рассылки, автоматически пересылать

информацию по определенной теме или отправлять информацию конкретным лицам по мере необходимости. Большинство почтовых программ предоставляют функции настройки, которые вы можете адаптировать к типу выполняемой работы, объему ежедневных сообщений электронной почты и потребностям работника. Эффективное использование электронной почты может снизить чрезмерную зависимость от личных консультаций и встреч.

Перемещение сообщений с физической почты на электронную почту может привести к значительной экономии почтовых расходов, расходных материалов для доставки и ресурсов сотрудников. Предприятия также могут снизить затраты на поддержку обслуживания клиентов, сосредоточив внимание на вариантах поддержки клиентов по электронной почте, а не на услугах, предоставляемых по телефону.

Современная электронная почта стала чем-то большим, чем просто текстовые сообщения. Вы можете создавать красочные, привлекательные листовки, используя тот же язык гипертекстовой разметки (HTML), который используется для создания веб-сайтов. Вложения электронной почты обеспечивают быстрый и удобный способ обмена файлами формата PDF, электронными таблицами и практически любыми компьютерными файлами размером.

1 Микросервис для автоматизации информирования сотрудников организации

1.1 Анализ предметной области

Зачастую в бизнесе необходимо делать информационные рассылки всем или определенной группе сотрудников. Причины для отправки сообщения электронной почты могут быть разные: предложение с ограниченным сроком действия, большое объявление или что-то еще, о чем сотрудники должны знать.

Все сегодняшние крупные почтовые сервисы могут справиться с обычной рассылкой без каких-либо трудностей. Но если появилась необходимость отправлять однотипные письма-напоминая ежедневно в определенный час, то начинают возникать проблемы. Например, напоминание сотрудникам в 9:45, что у них скоро начнется ежедневный митинг. Здесь функционала почтовых сервисов может не хватить. Одним из решений данной задачи – это заставлять каждый раз почтового администратора делать данную рассылку, что естественно неудобно. Потому что у разных команд может начинаться в разное время свои митинги, а также само количество команд может быть достаточно большим. Причем сам почтовый администратор может по разным причинам забыть сделать рассылку.

Популярные почтовые сервисы *Gmail*, *Mail.ru* и *Yandex* имеют отложенную отправку письма. Благодаря этой функциональности можно запланировать автоматическую отправку письма в выбранный день и время. Она имеет данные преимущества:

- если необходимо отправить сообщение поздно вечером, но, чтобы получатель получил его в рабочие часы;

- если необходимо спланировать поездку или не будет доступа к интернету, но важно, чтобы рассылка гарантированно дошла получателям;

- до тех пор, пока отложенное письмо не отправлено, можно отредактировать его тему, содержание послания.

Однако данный функционал не решает поставленную задачу из-за некоторых ограничений:

- в каждом сервисе есть ограниченное создание отложенных писем;

- неавтоматизированная работа. Администратору придется вручную создавать сотни писем на каждый рабочий день;

3) нет удобного *UI-интерфейса* для управления рассылкой, редактирования получателей;

4) нет статистики отправленных сообщений и статистики о количестве сотрудников в виде удобных графиков;

5) нет *API*, по которому можно получить всю информацию.

С автоматической отправкой сообщений может справиться *Microsoft Outlook* - персональный информационный менеджер с функциями почтового клиента. В данном менеджере можно создать макрос, который будет отправлять письма по расписанию. Рассмотрим плюсы этого метода:

1) макрос в заданный интервал времени (раз в сутки, в среду вечером, раз в неделю или в иное указанное вами время) отправляет письма из указанной папки *Outlook*;

2) данные письма не удаляются, а служат своего рода шаблоном;

3) один раз создать шаблон, один раз настроить задачу в *Outlook*.

Минусы:

1) *Outlook* для бизнеса стоит 1440 рублей на одного пользователя в месяц. Для многих кампаний — это очень большая сумма;

2) чтобы написать макрос, нужно знать язык макросов *Outlook*;

3) нет удобного *UI-интерфейса* для управления рассылкой, редактирования получателей;

4) нет статистики отправленных сообщений и статистики о количестве сотрудников в виде удобных графиков;

5) в праздничные дни письма будут приходить и засорять почтовый ящик, так как нельзя сверить с официальным рабочим календарем.

Практически со всеми проблемами выше справляется платформа *UniSender* - это простой в использовании сервис *email-маркетинга*, где каждый пользователь имеет доступ ко всем функциям и инструментам, а также круглосуточную поддержку клиентов через чат, электронную почту и телефон.

С помощью *Unisender* можно создать почтовую кампанию менее чем за 15 минут. Платформа имеет широкий спектр услуг. Можно удобно импортировать контакты для будущих кампаний, а также удобно создавать список с нуля. Также можно отправлять и отслеживать результаты. Платформа составляет подробный аналитический отчет. Отчеты показывают поставки, открытия, клики и отписки. Можно узнать какие части электронной почты создают наибольшую конверсию, сравнить, какие электронные письма работают лучше, и узнайте, как кампания работала с течением времени. Письма по расписанию для этой платформа реализованы с помощтю удобного пользовательского интерфейса. В интерфейсе есть кнопки для выбора частоты отправки (еженедельно, ежемесячно), поле для времени, конкретного числа. Также можно нажать кнопку отправки в последние дни месяца Ко всему этому данная платформа имеет очень красивый и современный сайт, по которому легко ориентироваться.

По итогу можно сказать, что *Unisender* - хорошее решение для выполнения нашей задачи, но у него есть ряд недостатков:

1) *Unisender* платный;

2) в праздничные дни письма будут приходить и засорять почтовый ящик, так как нельзя сверить с официальным рабочим календарем;

3) множество лишних технологий ненужных для реализации нашей задачи, за которые надо платить.

Из всех существующих технологий, на данный момент в открытом доступе нет бесплатного и гибкого ПО для автоматизированной отправки сообщений по расписанию.

1.2 Постановка задачи

Необходимо разработать микросервис, который будет автоматически по расписанию отправлять сообщение в определенное время сотрудникам, добавленным в рассылку.

Микросервис должен удовлетворять следующим требованиям:

- перед рассылкой необходимо сверять сегодняшний день с официальным производственным календарем;

- необходимо реализовать систему аутентификации и авторизации, где каждый пользователь имеет роли. Пользователи с ролью *«User»* должны иметь ограниченные права, позволяющие получать только список всех сотрудников и находить конкретного сотрудника. Пользователям с ролью *«Admin»* доступен весь функционал микросервиса;

- интерфейс нужно реализовать в виде сайта, в котором нужно реализовать:

1) просмотр таблицы пользователей или сотрудников;

2) добавление, удаление и редактирование сотрудников и пользователей, где для редактирования и добавления открываются дополнительное окна;

3) возможность посмотреть или обновить текущее сообщение, которое будет отправляться сотрудникам, и время;

4) график, показывающий количество сотрудников и количество отправленных сообщений каждый месяц текущего года;

5) отображение доступных *API* микросервиса с описанием входных параметров, возможных кодов ошибок и примерами тела запроса, если оно необходимо;

6) отдельная форма авторизации.

- сделать дополнительное обращение к микросервису по *REST API* с помощью *JWT*, в заголовке которого должно содержаться название алгоритма, используемого для шифрования, а в секции «Полезная нагрузка» логин, список ролей, дата выдачи *JWT* и дата его истекания. Методы *REST API*:

1) добавление пользователя и сотрудника;

2) удаление пользователя и сотрудника;

3) получение списка пользователей и сотрудников;

4) получение информации о пользователе (идентификатор, логин, пароль, список ролей, электронный адрес почты) по логину;

5) получение информации о сотруднике (идентификатор, имя, фамилия, электронный адрес почты) по почте и идентификатору;

6) удаление пользователя и сотрудника;

7) обновление данных пользователя и сотрудника;

8) получение информации сообщения, отправляемое сотрудникам, и времени;

9) обновление отправляемого сообщения и времени;

10) получение информации о статистике за все время или за конкретный год;

11) получение *jwt*.

1.3 Архитектура приложения

1.3.1 Микросервисная архитектура

Программной обеспечение будет реализовано в виде микросервиса, на сегодняшний день это самая удобная и современная реализация.

Микросервисы - это архитектурный и организационный подход к разработке программного обеспечения, при котором ПО состоит из небольших независимых сервисов, которые взаимодействуют через четко

определенные *API*. Эти сервисы принадлежат небольшим автономным командам.

Архитектуры микросервисов облегчают масштабирование и ускоряют разработку приложений, позволяя внедрять инновации и ускоряя вывод на рынок новых функций.

Характеристики микросервисов:

- автономность. Каждая компонентная служба в архитектуре микросервисов может разрабатываться, развертываться, эксплуатироваться и масштабироваться без влияния на функционирование других служб. Службы не должны делиться своим кодом или реализацией с другими службами. Любая связь между отдельными компонентами происходит через четко определенные *API*;

- каждый сервис рассчитан на набор возможностей и ориентирован на решение конкретной задачи. Если разработчики добавляют больше кода в службу с течением времени, и служба становится сложной, ее можно разбить на более мелкие службы.

Преимущество микросервисов:

- гибкость. Микросервисы способствуют организации небольших независимых команд, которые ответственно за свои услуги. Команды действуют в небольшом и хорошо понятном контексте и имеют возможность работать более независимо и быстрее. Это сокращает время цикла разработки. Вы получаете значительную выгоду от совокупной пропускной способности организации;

- масштабируемость. Микросервисы позволяют независимо масштабировать каждую службу для удовлетворения спроса на поддерживаемую ею функцию приложения. Это позволяет командам правильно определять потребности инфраструктуры, точно измерять стоимость функции и поддерживать доступность, если на услугу наблюдается всплеск спроса;

- простота развертывания. Микросервисы обеспечивают непрерывную интеграцию и непрерывную доставку, что позволяет легко пробовать новые идеи и откатываться, если что-то не работает. Низкая стоимость отказа позволяет экспериментировать, упрощает обновление кода и ускоряет вывод на рынок новых функций;

- технологическая свобода. Архитектуры микросервисов не следуют подходу «один размер подходит всем». Команды имеют свободу выбора лучшего инструмента для решения своих конкретных проблем. Как следствие, команды, создающие микросервисы, могут выбрать лучший инструмент для каждой работы;

- многоразовый код. Разделение программного обеспечения на небольшие, четко определенные модули позволяет командам использовать функции для различных целей. Сервис, написанный для определенной функции, может использоваться в качестве строительного блока для другой функции;

- сочетание технологий. Команды могут выбрать технологию, которая лучше всего подходит для их службы, используя сочетание технологических стеков по мере необходимости;

- изоляция данных. Гораздо проще выполнять обновления схемы, поскольку затрагивается только один микросервис. В монолитном приложении обновление схемы может стать очень сложным, поскольку разные части приложения могут касаться одних и тех же данных, что делает любые изменения схемы рискованными.

1.3.2 Шаблон *Controller-Service-Repository*

Шаблон *Controller-Service-Repository* распространен во многих приложениях *Spring Boot*. Этот паттерн используют, потому что он отлично справляется с разделением проблем. Уровень контроллера в верхней части этой картины несет полную ответственность за раскрытие функциональности, чтобы она могла потребляться внешними объектами

(включая, возможно, компонент пользовательского интерфейса). Слой работы с хранимыми данными (*Repository*) в нижней части этой картины отвечает за хранение и извлечение некоторого набора данных. Уровень сервиса - это то, куда должна идти вся бизнес-логика. Если бизнес-логика требует извлечения или сохранения данных, она подключается к слою данных. Если кто-то хочет получить доступ к этой бизнес-логике, они проходят через контроллер, чтобы добраться туда [1].

Это довольно простое разделение проблем. Если код связан с хранением или поиском, он должен идти в слой данных. Если речь идет о раскрытии функциональности, она входит в контроллер. Все уникальное в бизнес-логике будет идти на уровне сервиса. Уровень сервиса не заботится о том, как к нему обращаются, он просто выполняет свою работу, используя слой данных там, где это важно. И контроллер просто передает работу на уровень сервиса, поэтому он может оставаться «худым».

1.4 Инструменты разработки

1.4.1 Язык *Java*

Микросервис будет написан на *Java*, потому что *Java* - это стабильный, легко читаемый и популярный язык программирования среди разработчиков. Когда дело доходит до построения архитектуры микросервисов, язык программирования *Java* более полезен. Простой синтаксис аннотаций упрощает создание архитектуры микросервисов. Кроме того, *Java* - отличный вариант, поскольку он обеспечивает пользовательский интерфейс, подключение к внутренним ресурсам и компонентам модели.

Java обеспечивает большую ценность в удобочитаемости, когда дело доходит до работы со сложными системами.

1.4.2 *Spring Boot*

Основной библиотекой шаблонной разработки, поддерживающим построение архитектуры микросервисов, является *Spring Boot*.

Spring Boot позволяет легко создавать автономные приложения на основе *Spring*, которые вы можете "просто запустить".

Большинству приложений *Spring Boot* требуется минимальная конфигурация пружины.

Преимущества Spring Boot [2]:

1) упрощенное и бесконфликтное управление зависимостями;

2) быстрота настройки и запуска автономных веб-приложений и микросервисов за гораздо меньшее время;

3) простая сборка артефактов, которые поставляется со встроенным сервером приложений *Tomact*, *Jetty* или *Undertow*;

4) *Spring Boot* предоставляет *API* для доступа к внутренним данным приложения, таким как подробные показатели, внутренняя работа приложения, состояние работоспособности;

5) никаких конфигураций на основе *XML*. Очень упрощенные свойства. Классы, которые управляются контейнером *Spring* инициализируются, настраиваются и подключаются автоматически.

1.4.3 СУБД *Spring JDBC*

Список пользователей и их данные будут храниться в базе данных. Для обращения к базе мы должны использовать технологию *Spring JDBC*. *JDBC* расшифровывается как *Java Database Connectivity*.

JDBC API можно предоставляет доступ к табличным данным, хранящимся в любой реляционной базе данных, а также позволяет сохранять, обновлять, удалять и извлекать данные из базы данных.

1.4.4 *MySQL*

В качестве СУБД в микросервисе используется *MySQL*. *MySQL* - это система управления реляционными базами данных основанная на запросах SQL с открытым исходным кодом.

Плюсы MySQL:

- поскольку *MySQL* с открытым исходным кодом, его можно использовать бесплатно;

- портативность. *MySQL* совместим со с многими операционными системами, такими как *Linux*, *Windows* и другими;

- бесшовное подключение. Для подключения к серверу *MySQL* доступны различные безопасные и бесшовные механизмы подключения;

- *MySQL* быстрый и надежный благодаря своей уникальной архитектуре механизма хранения данных. Он обеспечивает очень высокие результаты по сравнению с другими базами данных, не теряя при этом существенной функциональности программного обеспечения. Он имеет быстрые утилиты загрузки из-за различной кэш-памяти.

Недостатки:

- *MySQL* не поддерживает очень большой размер базы данных так же эффективно;

- *MySQL* не очень эффективно обрабатывает транзакции и подвержен повреждению данных;

- *MySQL* не имеет хорошего инструмента разработки и отладки по сравнению с другими базами данных;

- *MySQL* не поддерживает ограничения проверки *SQL*.

1.4.5 *HikariCP*

Для более эффективной работы микросервиса необходимо обратить внимание на пул соединений к базе данных. Пул соединений - это метод, используемый для повышения производительности в приложениях с динамическим контентом, управляемым базой данных. Открытие и закрытие соединений с базой данных может показаться не дорогостоящим расходом, но он может складываться довольно быстро. Предположим, что для установления соединения требуется 5 мс и 5 мс для выполнения вашего запроса (полностью составленные числа), 50% времени-это установление

соединения. Расширьте это до тысяч или десятков тысяч запросов, и будет много потраченного впустую сетевого времени.

Причины, по которой используют пулы соединений:

- постоянное открытие и закрытие соединений может быть дорогостоящим. Кэш и повторное использование;

- когда активность резко возрастает, вы можете ограничить количество подключений к базе данных. Это заставит код блокировать до тех пор, пока соединение не будет доступно. Это особенно полезно в распределенных средах;

- разделите общие операции на несколько пулов.

HikariCP - это очень быстрый легкий пул соединений *Java*. *API* и общая кодовая база относительно невелики и высоко оптимизированы. Он также не срезает углы для производительности, как многие другие реализации пула соединений *Java*.

1.4.6 *REST API*

Чтобы взаимодействовать с микросервисом из браузера, других микросервисов или систем, важно осуществить набор определений и протоколов для создания и API. *API* будет соответствовать архитектурному стилю *REST*. *REST* - это архитектурный стиль для обеспечения стандартов между компьютерными системами в Интернете, облегчающий взаимодействие систем друг с другом.

RESTful API - это *API*, который следует этим правилам, обеспечивая гибкость, пропускную способность и преимущества скорости для интеграции программного обеспечения и коммуникаций.

REST API служат посредником между вашим клиентом и вашим хост-сервером, отправляя запросы ресурсов и доставляя ответы через *HTTP*. В дизайне *REST* клиент инициирует взаимодействие - например, доступ к веб-странице через URL-адрес является обычно используемым HTTP-запросом.

Наиболее распространенными методами HTTP являются:

- *GET*: для получения ресурса;

- *POST*: для создания нового ресурса;

- *PUT*: для обновления ресурса;

- *DELETE*: для удаления ресурса.

1.4.7 *Spring Security*

Для реализации безопасности микросервиса используется *Spring security*.

Spring security обеспечивает аутентификацию и авторизацию с помощью простых фильтров. Веб-приложения подвержены угрозам безопасности и атакам, так как они доступны любому пользователю Интернета [3]. Могут существовать некоторые конечные точки *REST*, имеющие ограниченный доступ к определенным пользователям, например, обновление записей или операции, связанные с администратором.

Мы можем использовать *Spring Security* для защиты *URL-адресов*. *Spring Security* - это платформа безопасности, которая защищает корпоративные приложения, предоставляя мощные настраиваемые функции безопасности.

Spring Security работает над следующими тремя основными концепциями

- аутентификация;

- авторизация;

- хранение паролей. *Spring Security* предоставляет несколько классов для их хеширования;

- фильтры *HTTP-запросов*.

1.4.8 Стандарт *JWT*

Чтобы дополнительно обезопасить микросервис, механизм аутентификации использует *JWT*.

JWT - это механизм проверки владельца некоторых данных JSON. Это закодированная *URL-безопасная* строка, которая может содержать неограниченное количество данных и криптографически подписана [2].

Идея работы такова:

- клиент заходит на сервер аутентификации с помощью логина и пароля;

- сервер создает *JWT* и отправляет его клиенты;

- при запросе к *API* приложения, клиент добавляет *JWT*;

- приложение по переданному *JWT* проверяет является ли пользователь тем, за кого себя выдает.

JWT состоит из 3-х частей: заголовок, полезные данные и подпись. Заголовок содержит данные о том, как должна шифроваться подпись. Полезные данные содержат любую информацию о пользователе, которую мы хотим передать (его логин, роли и т.д.), а также время жизни *JWT*. Подпись создается по следующему алгоритму:

1) с помощью *base64* кодируется заголовок и полезная информация созданные ранее;

2) затем происходит конкатенация полученных строк через точку;

3) полученная строка хешируется алгоритмом, прописанным в заголовке, на основе секретного ключа.

1.4.9 Инструмент *Maven*

Для создания исполняемого приложения из исходного кода необходим инструмент сборки. Он также необходим для таких процессов, как генерация исходного кода, генерация документации исходного кода, компиляции исходного кода, упаковка скомпилированных кодов в файлы *JAR* или *Docker*. Одним из самых удобных и популярных инструментов является *Maven* [4].

Maven - это популярный инструмент сборки с открытым исходным кодом, разработанный группой *Apache* для создания, публикации и

развертывания нескольких проектов одновременно для лучшего управления проектами.

Основываясь на объектной модели проекта (*POM*), этот инструмент упростил жизнь разработчиков *Java* при разработке отчетов, проверке сборки и настройке автоматизации тестирования.

Maven загружен многими ценными и полезными функциями, которые во многом объясняют его популярность. Вот некоторые из наиболее примечательных функций *Maven*:

- огромный, постоянно растущий репозиторий пользовательских библиотек;

- возможность легко настраивать проекты, используя лучшие практики;

- управление зависимостями с автоматическим обновлением;

- обратная совместимость с предыдущими версиями;

- сильная отчетность об ошибках и целостности;

- автоматическое управление родительскими версиями;

- обеспечивает согласованное использование во всех проектах;

- он расширяется, и вы можете легко писать плагины, используя языки сценариев или *Java*.

Maven настолько полезен благодаря объектной модели проекта (*POM*), которая представляет собой *XML-файл*, содержащий всю информацию о проекте и деталях конфигурации. *POM* содержит описание проекта, детали, касающиеся управления версиями и управления конфигурацией проекта [5].

Maven в основном используется для проектов на основе *Java*, помогая загружать зависимости, которые относятся к библиотекам или *JAR-файлам*. Инструмент помогает получить правильные файлы *JAR* для каждого проекта, так как могут быть разные версии отдельных пакетов.

Благодаря *Maven* загрузка зависимостей не требует посещения официальных сайтов различных программ. Инструмент также помогает создать правильную структуру проекта.

1.5 Реализация

В данной работе реализован архитектурный паттерн *«Controller-Service-Repository»*, который разделяет микросервис на слои.

В соответствии с данным паттерном разработку микросервиса можно разделить на следующие этапы:

- Разработка слоя хранения данных;
- Разработка сервисов;
- Разработка контроллеров;

1.5.1 Слой хранения данных

В работе было создано 4 таблицы: *«employee», «app_user», «user_role», «settings», «statistics»*. Логическая модель базы данных представлена на рисунке 1.1.

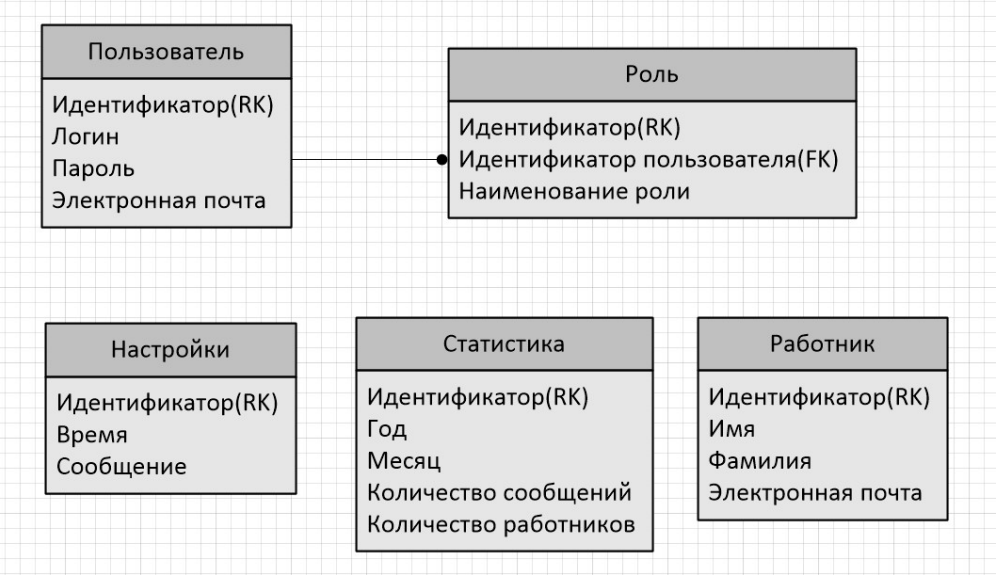


Рисунок 1.1 - Логическая модель базы данных

Таблица *«employee»* представляет сотрудников организации, которым будет отправляться сообщения. Таблица хранит персональные данные, такие как имя, фамилию, адрес электронной почты. Ее структура описана в таблице 1.1.

Таблица 1.1 – Структура таблицы «*employee*»

Название поля	Описание поля	Тип данных
id	Уникальный идентификатор	*Varchar*
firstName	Имя	*Varchar*
lastName	Фамилия	*Varchar*
email	Электронная почта	*Varchar*

Таблица 1.2 содержит описание структуры таблицы *«app_user»*, в которую передаются данные пользователей приложения. Таблица связана с таблицей *«user_role»*, по внешнему ключу, связь «один ко многим».

Таблица 1.2 – Структура таблицы *«app_user»*

Название поля	Описание поля	Тип данных
id	Уникальный идентификатор	*Varchar*
login	Логин	*Varchar*
password	Пароль	*Varchar*
email	Электронная почта	*Varchar*

В таблице *«user_role»* представлены роли пользователей. При удалении пользователя, все его роли также удаляются. Структура представлена в таблице 1.3.

Таблица 1.3 – Структура таблицы *«user_role»*

Название поля	Описание поля	Тип данных
id	Уникальный идентификатор	*Varchar*
role_type	Тип роли(*ADMIN*, *USER*)	*Varchar*
user_id	Идентификатор пользователя	*Varchar*

Отправляемое каждому сотруднику сообщение и время хранится в таблице *«settings»*. Ее структура описана в таблице 1.4.

Таблица 1.4 – Структура таблицы *«settings»*

Название поля	Описание поля	Тип данных
id	Уникальный идентификатор	*Varchar*
email_message	Сообщение	*Varchar*
departure_time	Время отправления сообщения	*Varchar*

Таблица *«statistics»* была создана для хранения статистики о количестве отправленных сообщений и сотрудников по месяцам. Структура данной таблицы показана в таблице 5.

Таблица 5– Структура таблицы «statistics»

Название поля	Описание поля	Тип данных
id	Уникальный идентификатор	*Varchar*
year	Сообщение	*Decimal*
months	Время отправления сообщения	*Decimal*
*number*_messages	Количество отправленных сообщений	*Decimal*
number_employess	Количество сотрудников	*Decimal*

Во время запуска микросервиса, автоматически запускаются подготовленные *SQL-скрипты*, которые создают соответствующие таблицы в базе данных, если их нет. Помимо создания таблиц, вставляется запись в таблицу *«settings»,* в которой находятся значения по умолчанию для поля *«email_message»* и *«departure_time»*.

Взаимодействие с базой данных происходит в 4 классах. Каждый класс имеет методы для взаимодействия с таблицами, такие как вставка, обновление, выборка и удаление.

Отдельно стоит отметить класс *«UderDaoImpl»*, который отвечает за взаимодействие с таблицы пользователей. В отличие от других таблиц, здесь существует связь «один ко многим» с таблицей ролей. Данная связь означает, что пользователь может иметь несколько ролей. При создании связи между таблицами было реализовано каскадное удаление, которое говорит о том, что при удалении родительского объекта, также будут удаляться все его дочерние объекты. Взаимодействие с таблицей пользователей происходит постоянно, а так как она еще и связана с другой таблицей, это дает дополнительную нагрузку на системные ресурсы микросервиса. При поиске клиента по логину или поиске всех клиентах была учтена *«n+1* проблема*»* [6]. Проблема с запросом *N + 1* возникает, когда ваш код выполняет *N* дополнительных операторов запроса для извлечения тех же данных, которые могли быть получены при выполнении основного запроса. Вместо двух отдельных запросов к таблице ролей и пользователей, был сделан один. Пример поиска всех пользователей представлен ниже:

```
SELECT user.id, user.login, user.email, user.password, role.role_type
FROM (SELECT id, login, email, password FROM app_user) AS user
LEFT JOIN user_role AS role ON user.id = role.user_id;
```

Реализация поиска пользователя по логину:

```
SELECT user.id, user.login, user.password, user.email, role.role_type
FROM (SELECT id, login, password, email FROM app_user HAVING login = ?) AS user INNER JOIN user_role AS role ON user.id = role.user_id;
```

Для обращение к базе данных используется класс «JdbcTemplate», который представлен в пакете *«org.springframework.jdbc.core»*. Данный класс облегчает использование *JDBC* и выполняет основной рабочий его процесс. Класс имеет много удобных методов, которые осуществляют SQL-запросы, помогают конвертировать приходящий результат в объекты java, а также ловить ошибки *JDBC* и представлять их в привычные исключения, представленные в пакете *«org.springframework.dao»*.

Физическая модель базы данных представлена на рисунке 1.2.

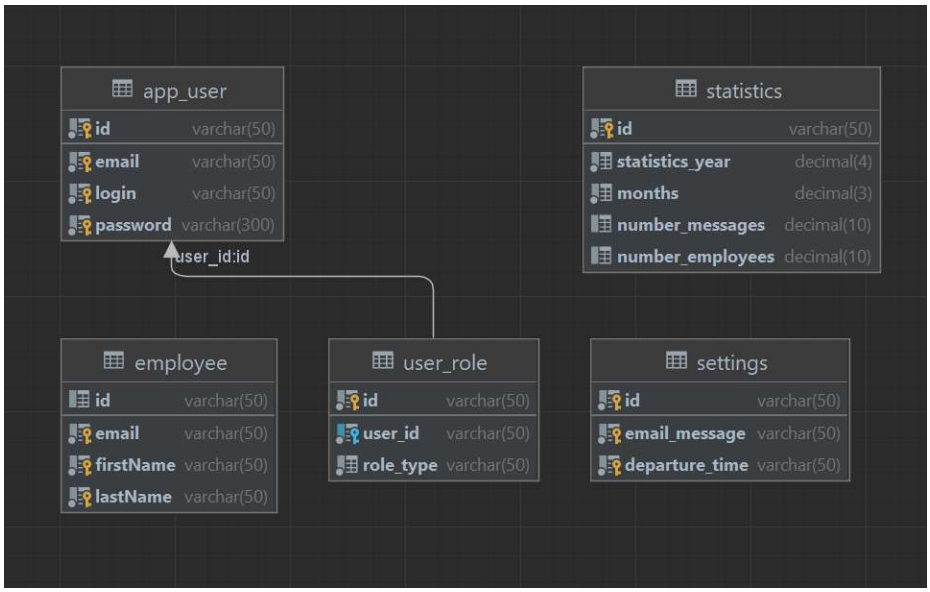


Рисунок 1.2 - Физическая модель базы данных

1.5.2 Слой сервисов

В слое сервисов реализован основной функционал микросервиса. Сервисы отвечают за валидацию данных и бизнес-логику, а также является связующим звеном между контроллером и слоем хранения данных.

Сервисы, методы которых выполняют CRUD-операции с сущностями пользователь, сотрудник, настройки и статистика, связаны непосредственно со слоем хранения данных. На их методы приходят на вход данные с контроллеров, которые они сначала валидируют. Валидация – это проверка данных на корректность. Зачастую это проверка на пустоту. Если данные не прошли проверку, то выбрасываются соответствующие ошибки. После валидации выполняется какая-либо логика, в том числе и обращение к слою хранения данных. Пример метода обновления данных в сервисе *«EmplyeeServiceImpl»* представлен ниже:

```
    @Override
    public void updateById(String id, String firstName, String
lastName, String email) {
```

```
        if       (StringUtils.isBlank(id))       throw       new
EmptyIdException();
        if     (StringUtils.isBlank(firstName))     throw     new
EmptyFirstNameException();
        if     (StringUtils.isBlank(lastName))     throw     new
EmptyLastNameException();
        if       (StringUtils.isBlank(email)       throw       new
EmptyEmailException();
        final Employee employee = employeeDao.findById(id);
        if      (employee      ==      null)      throw      new
EmployeeNotFoundException();
        employee.setFirstName(firstName);
        employee.setLastName(lastName);
        employee.setEmail(email);
        employeeDao.update(employee);
    }
```

На вход данного метода приходят 4 строки, которые мы проверяем на пустоту с помощью утилитного класса *«StringUtils»* библиотеки *«commons-collections4»* и его метода *isBlank*. Если какое-то поле пустое, то выбрасывается соответствующие исключение. Далее идет обращение к классу, работающему с базой данных для поиска работника, данные которого нужно обновить. После успешного нахождения в поля, найденного объекта, вставляются новые значения и затем происходит вызов метода обновления.

Подобная структура характерна и для методов вставки, поиска и удаления.

Помимо *CRUD-методов* сервисы берут на себя ответственность за выполнения всех основных алгоритмов микросервиса. За связь с почтовым сервером и отправкой сообщений отвечает сервис *«MailSenderService»*. Стоит отметить, что данный сервис взаимодействует также с пятью другими сервисами. Алгоритм единственного метода данного класса для отправки сообщения таков:

1) Вызов методов сервиса «CalsendarService» для скачивания рабочего календаря, если это важно, и проверка сегодняшнего дня на выходной;

2) обращение к сервису *«EmployeeService»* для поиска всех работников;

3) из сервиса *«SettingsService»* получить текст сообщение, которое необходимо отправлять;

4) с помощью класса *«JavaMailSener»* из пакета *«org.springframework.mail.javamail»* сформировать сообщение и отправить всем полученным сотрудникам на их электронную почту;

5) обновить статистику, вызвав метод нужный метод сервиса *«StatisticsService».*

Работа с официальным производственным календарем происходит в сервисе *«CalsendarService»*. В нем есть 2 метода. Первый метод отвечает за загрузка календаря в формате .csv с официального государственного сайта «Портал открытых данных Российской Федерации». Скачивание происходит каждый год 1 января или перед первым отправлением сообщений, если календарь не скачен. Второй метод проверяет является ли сегодняшний день выходным.

1.5.3 Слой контроллеров

Слой контроллеров отвечает за обработку входных запросов от пользователей и отправке пользователю результатов. Здесь не должно находится никакой бизнес-логик, изменений состояний модели или прямого обращения к базе данных.

В данном микросервисе есть 2 типа контроллеров. Первый тип отвечает за *REST-API*, который использует *HTTP-запросы* для получения, обновления, удаления и создания данных. *Spring* сильно упрощает написание данного функционала с помощью специальных аннотация. Если запрос направлен на создания данных, то нужно пометить его аннотацией *«@PostMapping»,* для поиска используется аннотация *«@GetMapping»,* для удаления *«@DeleteMapping»,* а при обновлении данных используется *«@PutMapping*». В аргументах данных аннотации прописывается унифицированный указатель ресурса – адрес ресурса в интернете, а также формат обмена данными. В данном микросервисе выбран *JSON*, он легче читается человеком и визуально красив. Если пользователь на вход передает тело, которое нам необходимо сконвертировать в *java объект*, то

используется аннотация *«@RequestBody»*. При несоответствии тела запроса с ожидаемым *java объектом*, возникнет исключение. При передаче информации в самом *url* запросе надо использовать аннотацию *«@PathVariable»*. Пример такого метода представлен ниже:

```
@Override
@DeleteMapping(value = "delete/{id}”)
    public void deleteUserById(
            @NotNull @PathVariable(name = "id") final String id
    ) {
         employeeService.deleteById(id);
    }
```

Здесь идентификатор пользователя передается внутри адреса запроса и вставляется в объект *«id»* класса *«String»*.

Пример метода, в который передается тело запроса:

```
@PutMapping(value = "/update/{id}")
    public void updateUser(
        @PathVariable("id") final @Nullable String id,
        @RequestBody final @Nullable Employee employee) {
        employeeService.updateById(id, employee);
    }
```

Запрос при таком методе должен соответствовать структуре:

```
{
  "id": "id",
  "email": "email@mail.ru",
  "firstName": "Oleg",
  "lastName": "Andreev"
}
```

Второй тип контроллеров отвечают за обработку данных, приходящих с интерфейса, а также за его отображения. Работы данных контроллеров сильно схожа с *REST* контроллерами. Отличие лишь в том, что они возвращают не *JSON*-ответ, а *HTML*-страницы. Для ее отображения используем класс *«ModelAndView»*. Он содержит поле *«viewName»*, в которое нужно передать путь к необходимой *HTML*, которая находится в ресурсах микросервиса. Также данный класс позволяет передавать объекты *Java*, значения полей которых можно внедрять в *HTML* с помощью сервисного механизма *Java-шаблонов – Thymeleaf*. Приме метода такого вида контроллера представлен ниже:

```
@Override
@PostMapping("/create")
public ModelAndView create(
    @ModelAttribute("user") @NotNull final Employee employee
) {
    employeeService.create(employee);
    return new ModelAndView("redirect:/employee");
}
```

1.5.4 Безопасность

Безопасность в микросервисе реализована с помощью Spring Security. Эта платформа позволяет построить систему безопасной авторизации и аутентификации. Алгоритм аутентификации:

1) При входе в систему пользователь вводит логин и пароль, которые участвуют при инициализации класса *«UsernamePasswordAuthenticationToken»*, который передается экземпляру классу *«AuthenticationManager»*;

2) *«AuthenticationManager»* вызывает метод для поиска пользователя из базы данных и делает сверку;

3) в случае если данные не сойдутся будет выброшено исключение;

4) если данные сошлись, то возвращается экземпляр класса *«Authentication»*;

5) далее для пользователя устанавливается контекст безопасности путем вызова метода *SecurityContextHolder.getContext().setAuthentication(...)*, в который передается объект, полученный из *«AuthenticationManager»*

Пароли пользователей хешируются с помощью *bcrypt* - криптографическая хеш-функция формирования ключа. Пользователь также имеет роли. В зависимости от роли пользователю предоставляются те или иный функции и возможности. Блок кода, в котором распределяются роли на адреса, представлен ниже:

```
http.authorizeRequests()
            .antMatchers("/images/**",  "/css/**",  "/fonts/**",
"/vendor/**", "/js/**").permitAll()

.antMatchers("/api/v1/authentication/login").permitAll()
```

```
.antMatchers("/api/v1/authentication/token").permitAll()
            .antMatchers("/").hasAnyAuthority (ADMIN, USER)

.antMatchers("/registration**").hasAnyAuthority(ADMIN)
            .antMatchers("/login").permitAll()

.antMatchers("/api/v1/employee").hasAnyAuthority(USER, ADMIN)

.antMatchers("/api/v1/employee/{.}").hasAnyAuthority(USER,
ADMIN)
            .antMatchers("/api/**").hasAnyAuthority(ADMIN);
```

Получение информации по *rest-api* происходит с помощью *JWT*. Получить его можно по *url-адресу*: */api/v1/authentication/token?username=test&password-test*, где вместо test нужно ввести свои пользовательские данные. Полученный *JWT* вставляется в заголовок любого запроса с ключом *«Authorization»* и значением *«Bearer {token}»*, где *«{token}»* - это полученный *JWT*. Все запросы, к которым необходим *JWT*, проходят через специальный фильтр, реализованный в классе *«JwtTokenFiltet»*. В нем изначально проверяется наличие заголовка с ключом *«Authorization»*, далее происходит валидация самого *JWT*. Если валидация пройдена, то пользователь будет авторизирован, а также получит запрашиваемые данные, иначе произойдет исключение.

1.6 Описание работы микросервиса

Чтобы пользоваться функциями приложения для начала необходимо пройти форму авторизации, представленной на рисунке 1.3.

Если нажать на кнопку *«Login»* при пустых полях ввода, появится предупреждение, представленное на рисунке 1.4.

После успешной авторизации пользователь перенаправляется на главную страницу сайта, где будет отображаться список сотрудников организации в виде таблицы, которая показана на рисунке 1.5.

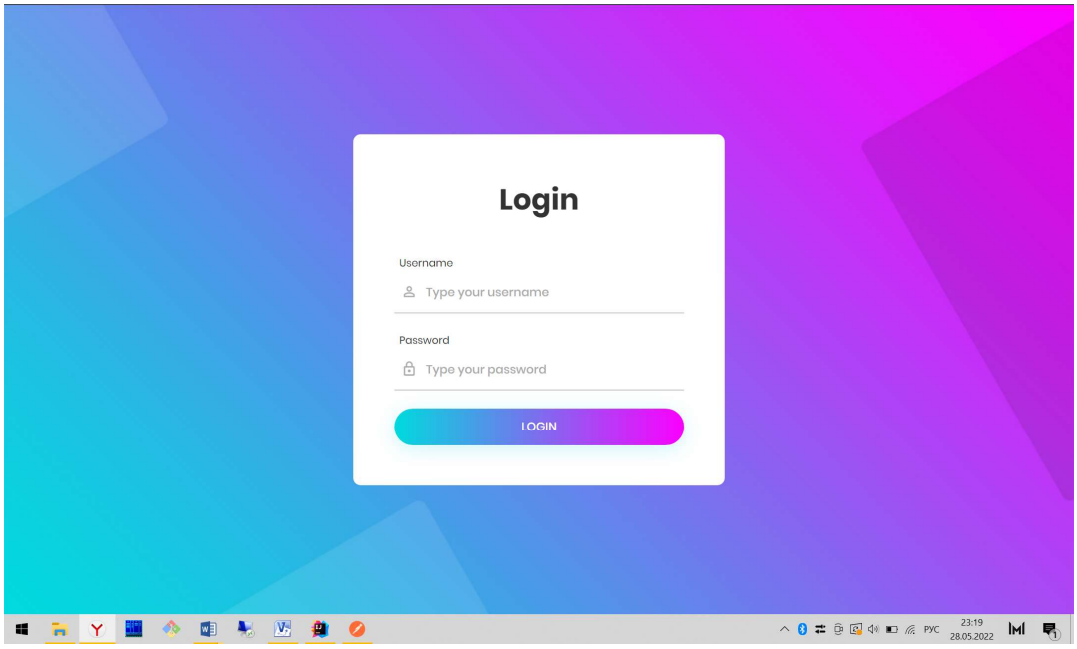


Рисунок 1.3 – Форма авторизации

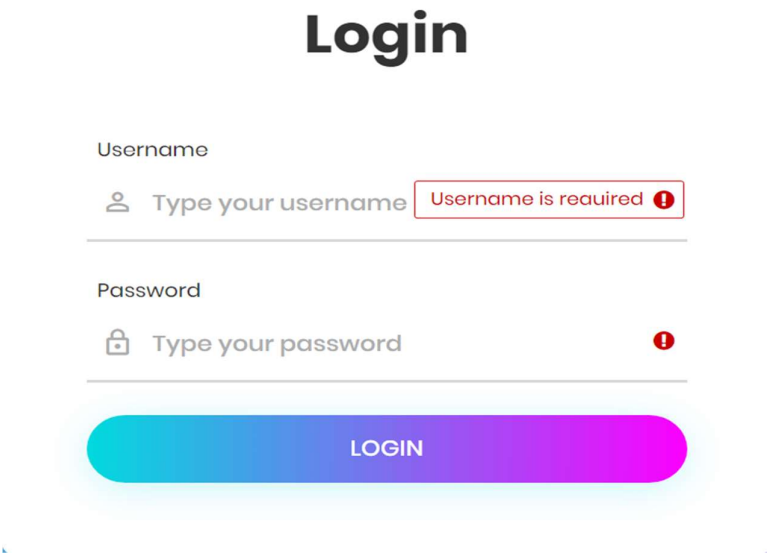


Рисунок 1.4 – Форма авторизации

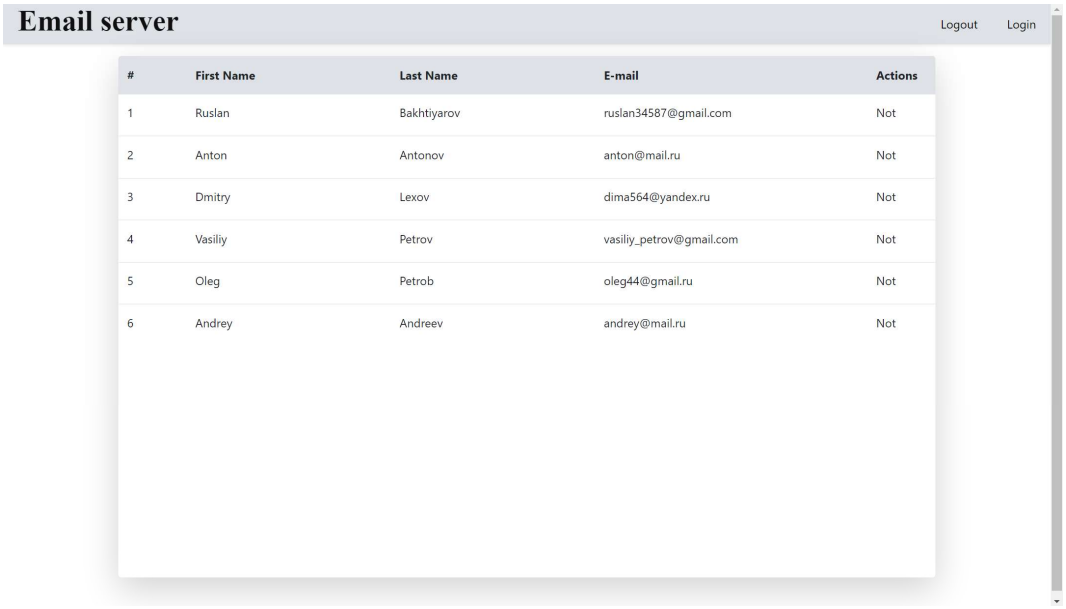


Рисунок 1.5 – Главная страница пользователя с ролью «*User*». Таблица сотрудников

В зависимости от роли пользователя на странице будут отображаться те или иные кнопки. Для пользователя с ролью «User» отображается только таблица сотрудников и кнопки *«Login»* и *«Logout»*. Для пользователя *с* ролью *«Admin»* помимо вышеупомянутых кнопок также появляются текстовые кнопки *«Employees»*, *«Users»*, *«Api»*, *«Add»*, *«Settings»*, *«Graph»* на верхней панели, а в колонке таблицы под названием *«Actions»* доступны кнопки редактирования и удаления в виде иконок. Данная страница представлена на рисунке 1.6.

При нажатии кнопки редактирования поверх таблицы появляется отдельная форма с полями ввода *«First Name»*, *«Last Name»* и *«Email»*. Форма показана на рисунке 1.7.

После изменении имени *Ruslan* на *Alexand*, представленном на рисунке 1.8, таблица также обновляется.

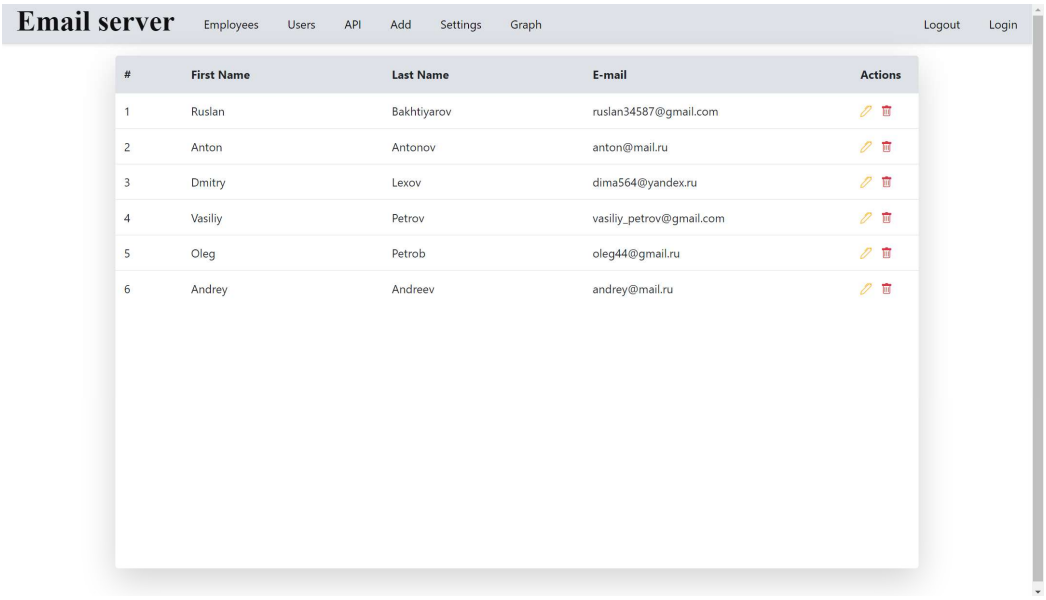


Рисунок 1.6 – Главная страница пользователя с ролью «Admin».

Таблица сотрудников

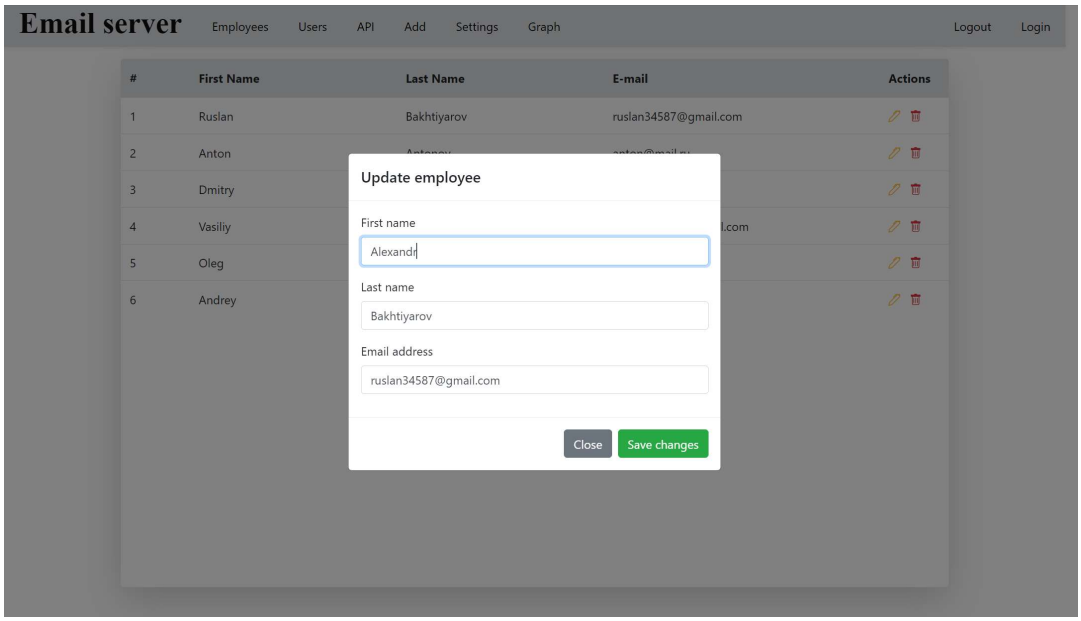


Рисунок 1.7 – Форма редактирования данных сотрудника

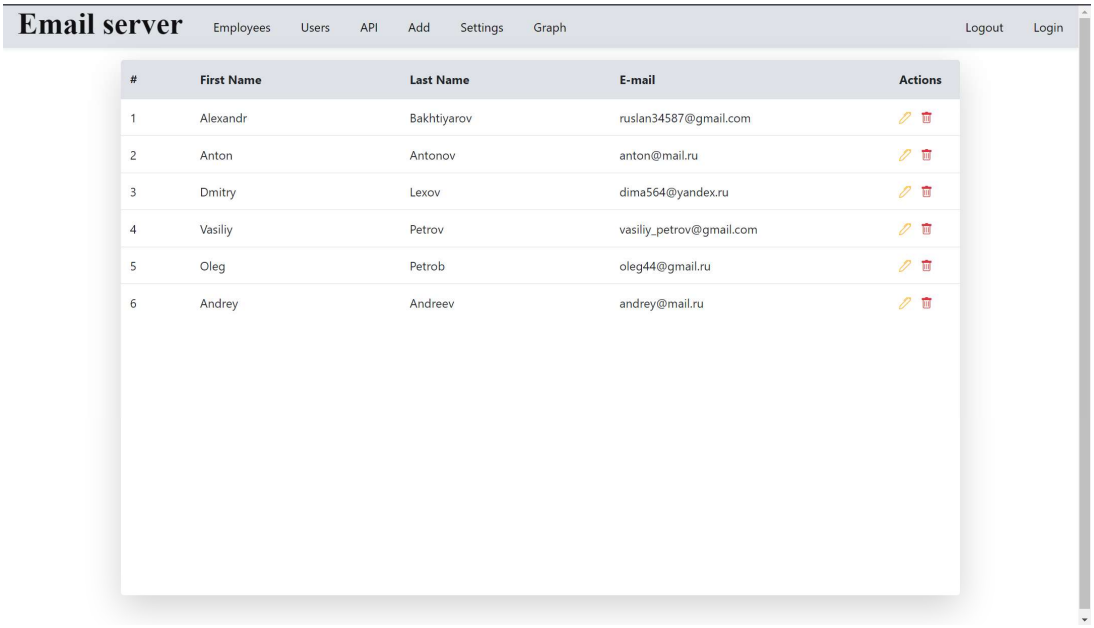


Рисунок 1.8 – Измененная таблица сотрудников после редактирования имени *Ruslan*

При нажатии иконки с корзиной напротив какого-либо сотрудника, удаляется соответствующий сотрудник. После удаления таблица моментально обновляется. Измененная таблица показана на рисунке 1.9.

Также администратор может добавлять новых сотрудников, для этого существует специальная кнопка *«Add»*, находящиеся над таблицей. После ее нажатия появляется похожая форма, как при редактировании. Форма приведена на рисунке 1.10. Результат добавление нового сотрудника можно увидеть на рисунке с номером 1.11.

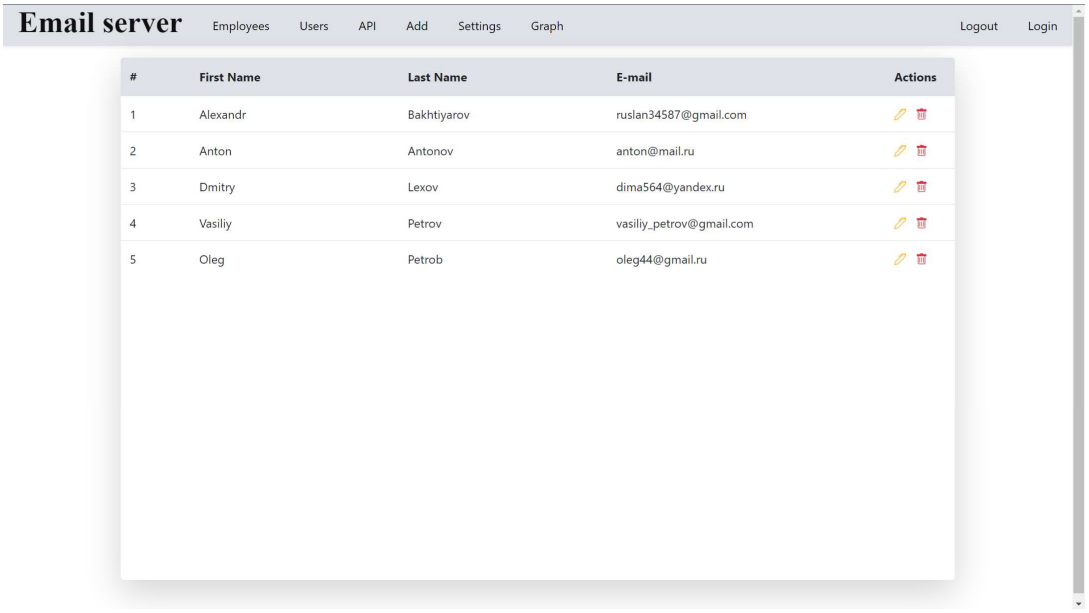


Рисунок 1.9 – Измененная таблица сотрудников после удаления сотрудника под номером 6

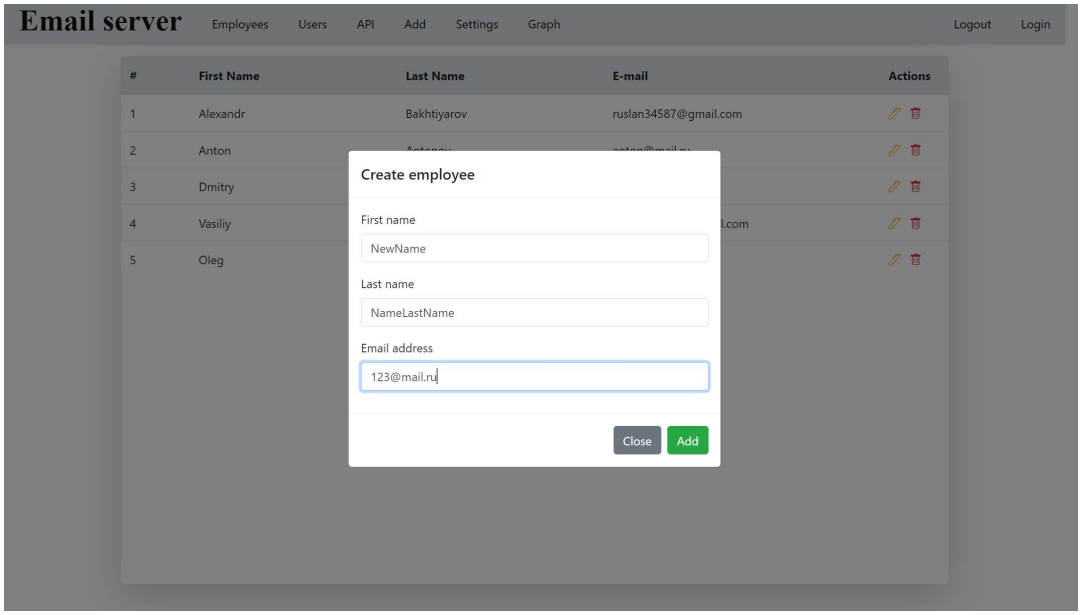


Рисунок 1.10 – Форма добавления сотрудника

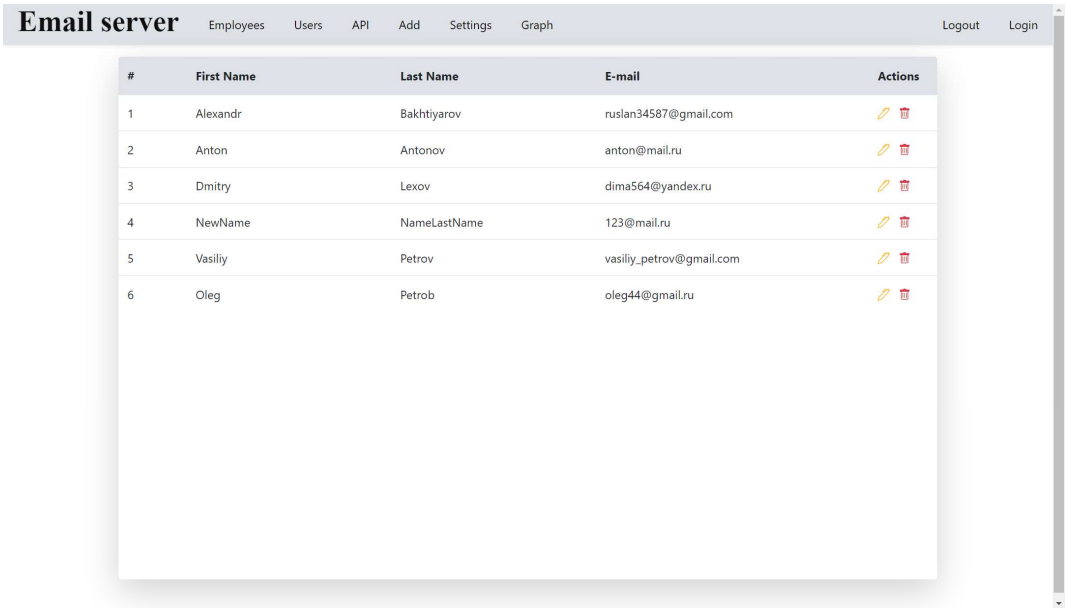


Рисунок 1.11 – Обновленная таблица после добавления нового сотрудника

Помимо таблицы с сотрудниками администратор может открыть таблицу с пользователями, нажав на кнопку *«Users»*. Таблица показана на рисунке 1.12.

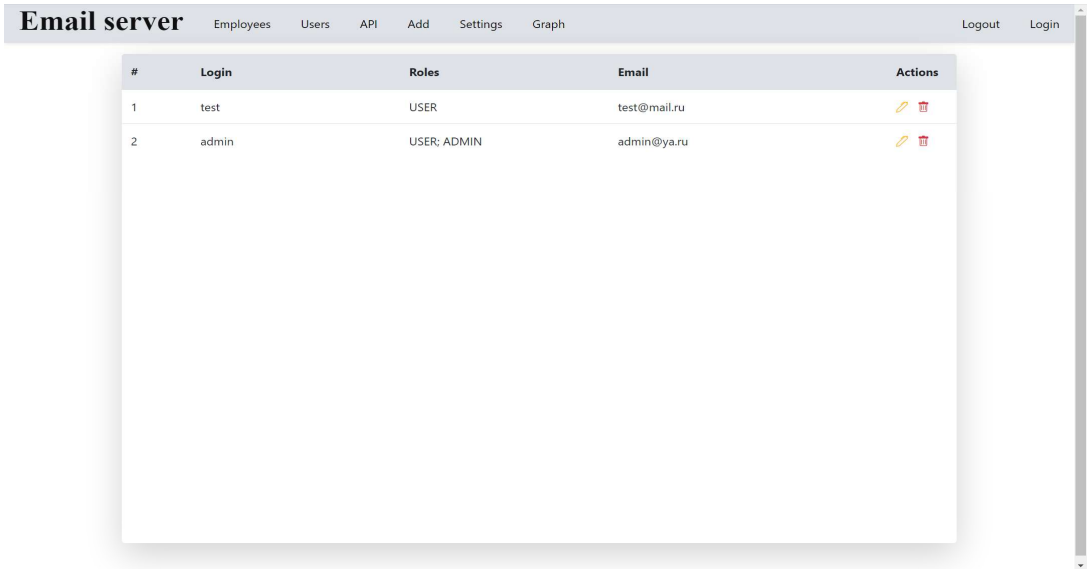


Рисунок 1.12 – Таблица пользователей

С таблицей пользователей доступны такие же действия как и с таблицей сотрудников. Изменяется лишь форма добавления, представленная на рисунке 1.13, где добавляются флажки с ролями. Добавленный пользователь показан на рисунке 1.14.

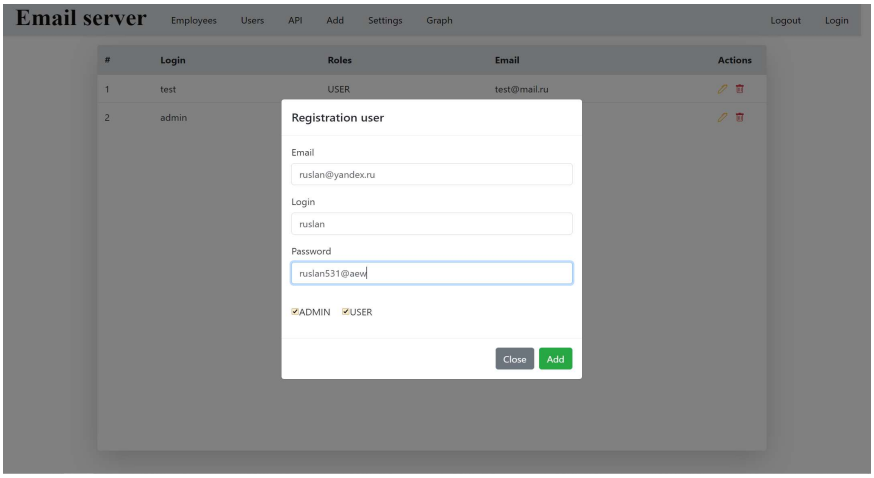


Рисунок 1.13 – Форма добавления нового пользователя

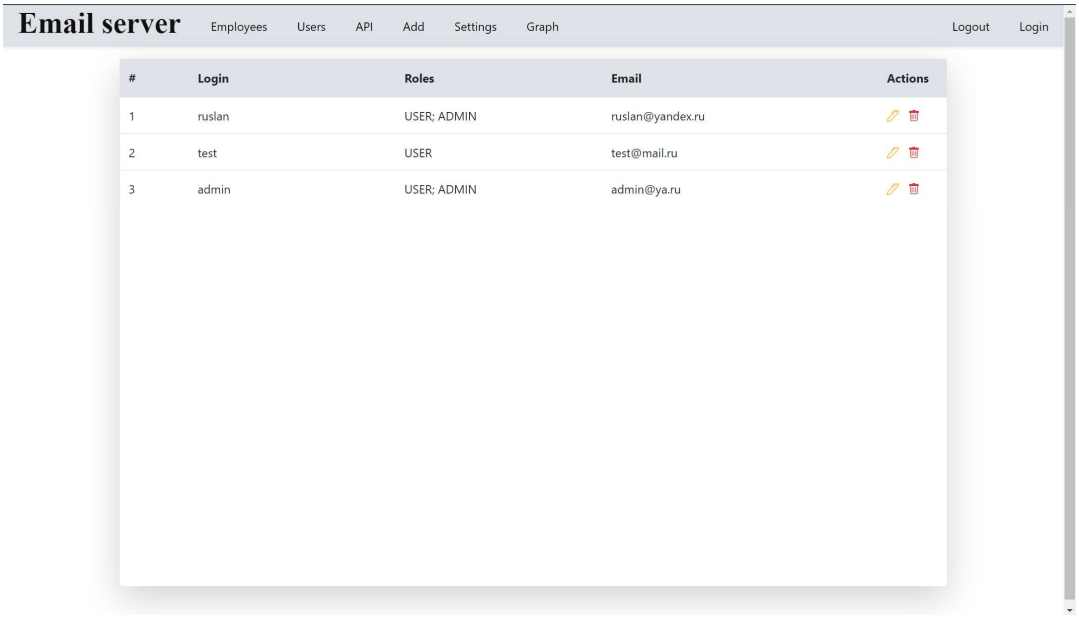


Рисунок 1.14 – Обновленная таблица после добавления нового пользователя

При нажатии на кнопку *«Settings»* появляется форма, на которой можно увидеть текущее поставленное время и сообщение. Их также можно поменять, для этого нужно ввести новые значения. Окно с настройками представлено на рисунке 1.15.

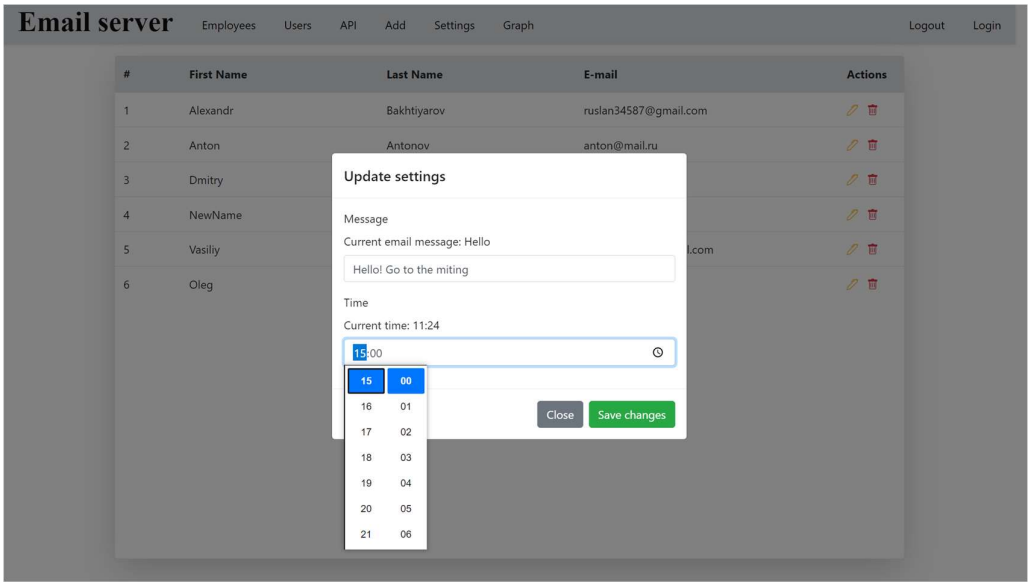


Рисунок 1.15 – Окно обновления настроек

При нажатии кнопки *«Graph»* появляется график, показывающий количество отправленных сообщений, а также количество сотрудников за каждый месяц текущего года. График изображен на рисунке 1.16.

Рисунок 1.16 – График

Если необходимо посмотреть только количество сотрудников, можно нажать кнопку *«Number of sms sent»*, чтобы убрать статистику о сумме сообщений, изображенная на рисунке 1.17. Или же наоборот, если нужно посмотреть только статистику сообщений, то надо нажать кнопку *«Number of employees»*.

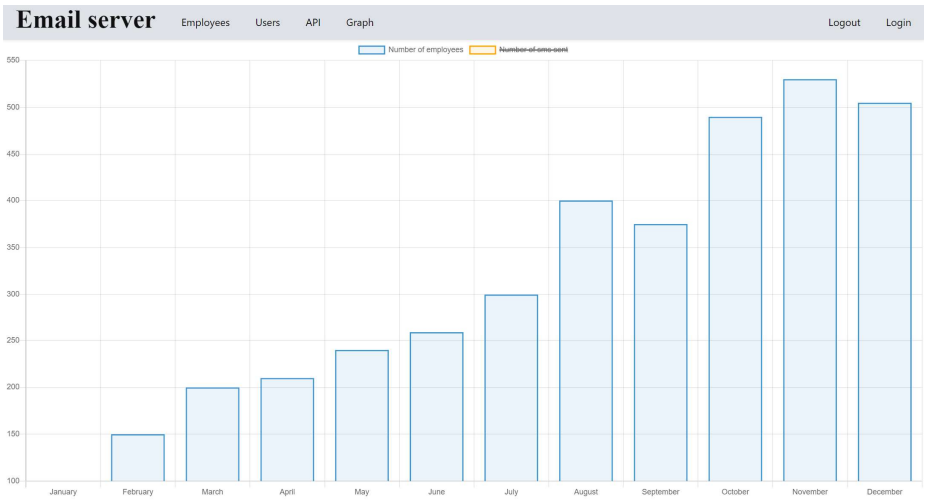


Рисунок 1.17 – График, отображающий количество сотрудников

Как уже было сказано, чтобы получать информацию по *API* важно получить *JWT*. Его получения показано на рисунке 1.18.

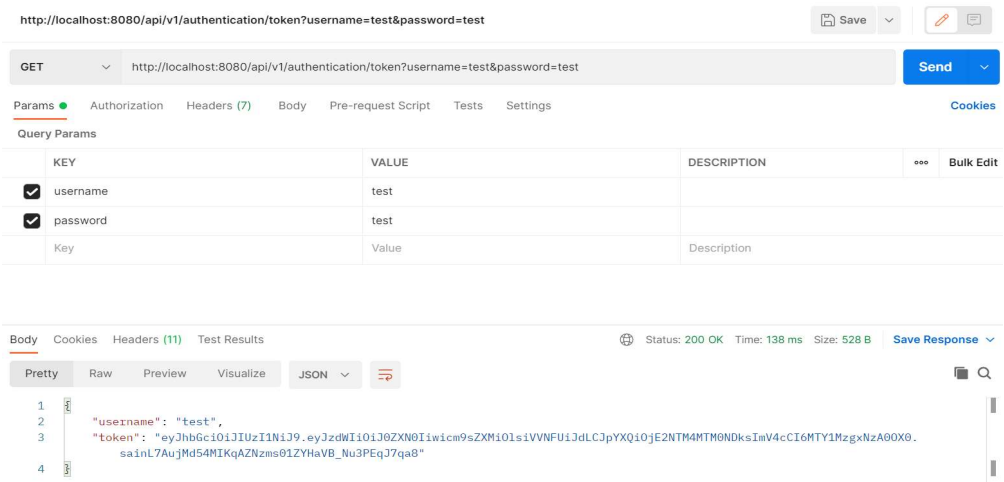


Рисунок 1.18 – Запрос на получение *JWT*

Полученный *JWT* вставляется в заголовок с ключом *«Authorization»*. Данный пользователь имеет роль *«User»*, а значит может получить только список сотрудник или получить конкретного сотрудника по идентификатору. Запрос представлен на рисунках 1.19 и 1.20.

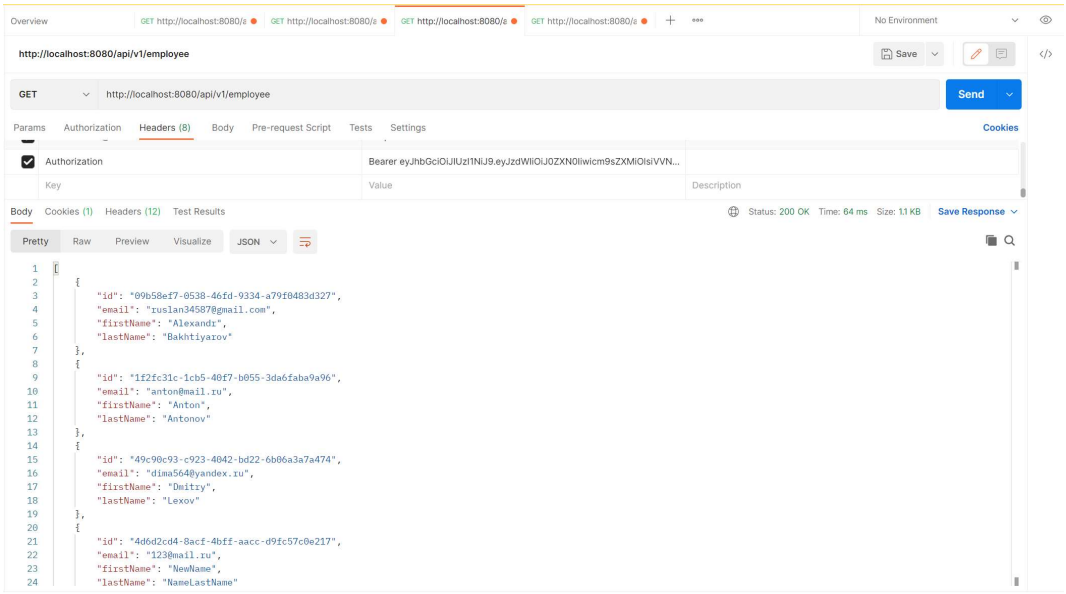


Рисунок 1.19 – Получение списка сотрудников организации

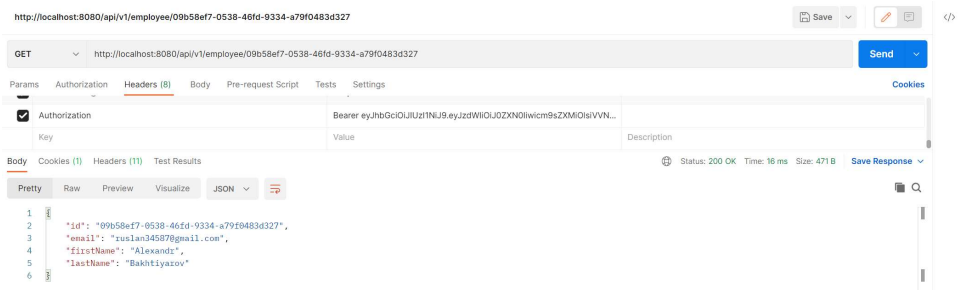


Рисунок 1.20 – Получение сотрудника по идентификатору

Если не предоставить JWT в заголовке запроса, то возникнет исключение, представленное на рисунке 1.21.

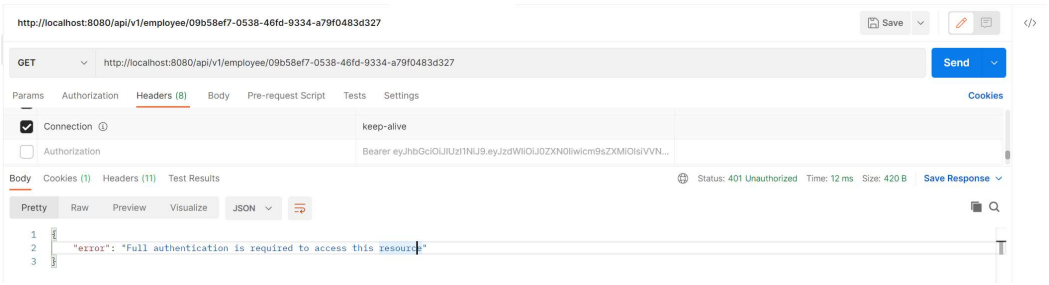


Рисунок 1.21 – Исключение при запросе с пустым заголовком

Если пользователь попытается получить данные, на которые у него нет доступа, то возникает исключение с ошибкой доступа, показанной на рисунке 1.22.

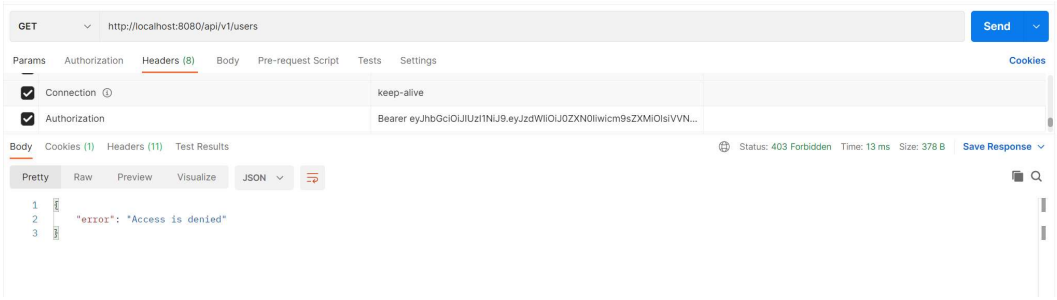


Рисунок 1.22 – Исключение при недостатке прав

Администраторы имеют полный доступ ко всем запросам. Поиск всех пользователей можно увидеть на рисунке 1.23.

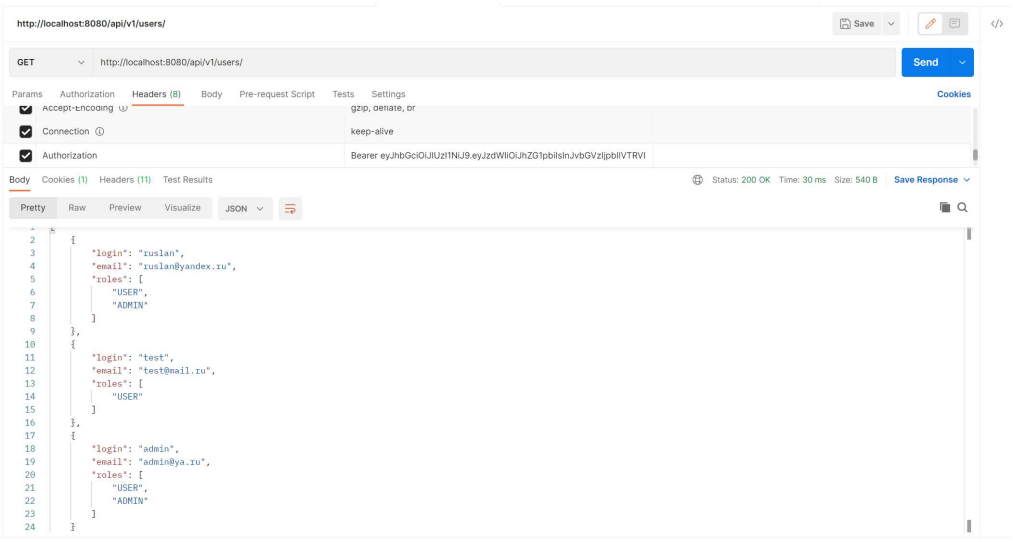


Рисунок 1.23 – Получение списка пользователей

Поиск пользователя по логину представлен на рисунке 1.24.

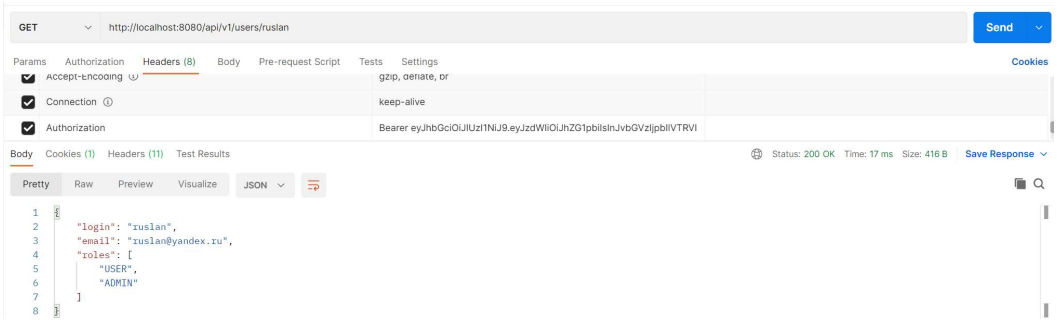


Рисунок 1.24 – Поиск пользователя по логину

Пользователя также можно удалить. Запрос удаление и положительный ответ изображен на рисунке 1.25.

Рисунок 1.25 – Запрос на удаления пользователя

Пример запросов на создание нового пользователя и сотрудника представлены на рисунках 1.26 и 1.27.

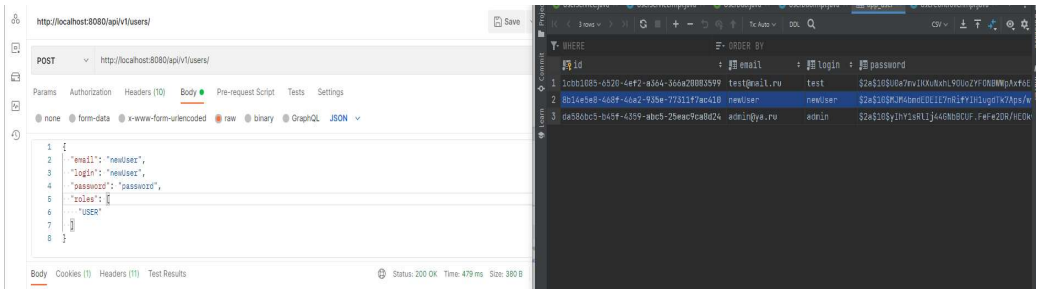


Рисунок 1.26 – Запрос на создание пользователя

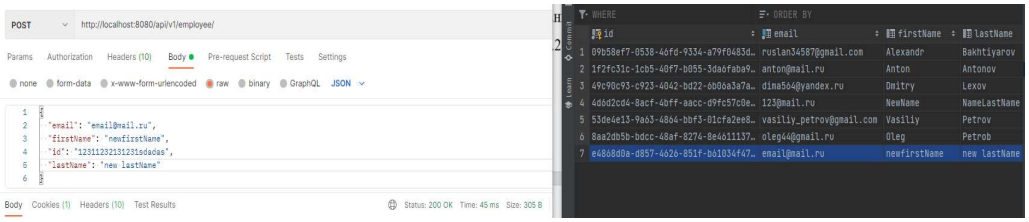


Рисунок 1.27 – Запрос на создание сотрудника

Сотрудника также можно найти по его электронной почте. Данный поиск продемонстрирован на рисунке 1.28.

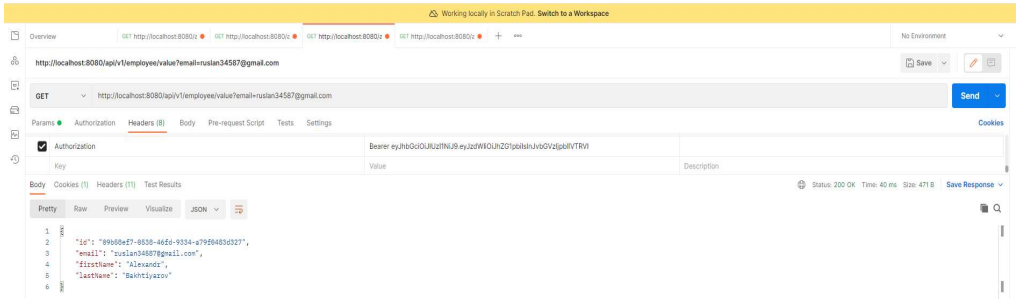


Рисунок 1.28 – Поиск сотрудника по электронной почте

Пример запроса на обновления данных сотрудника изображен на рисунке 1.29.

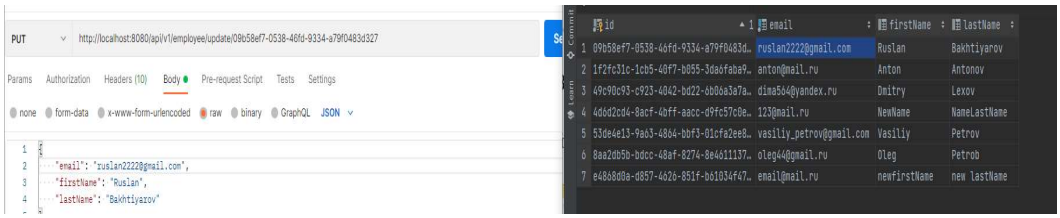


Рисунок 1.29 – Запрос на обновление данных сотрудника

Запросы на получение и изменение сообщения и времени можно увидеть на рисунках 1.30 и 1.31.

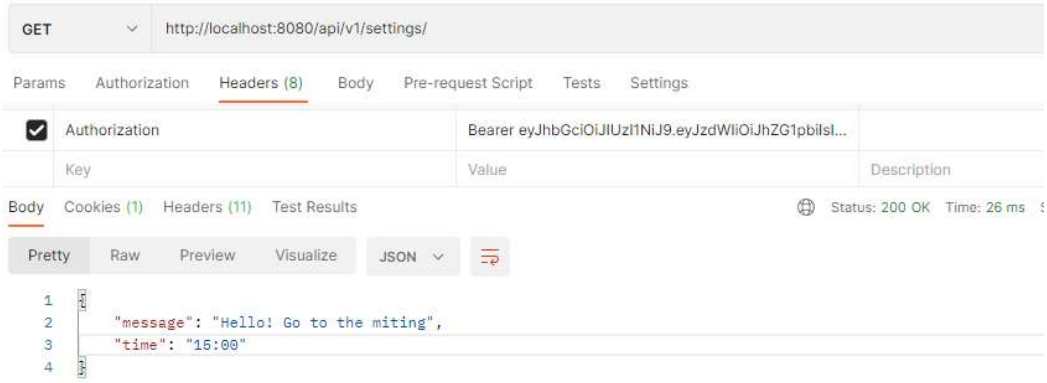


Рисунок 1.30 – Запрос на получения сообщения и времени

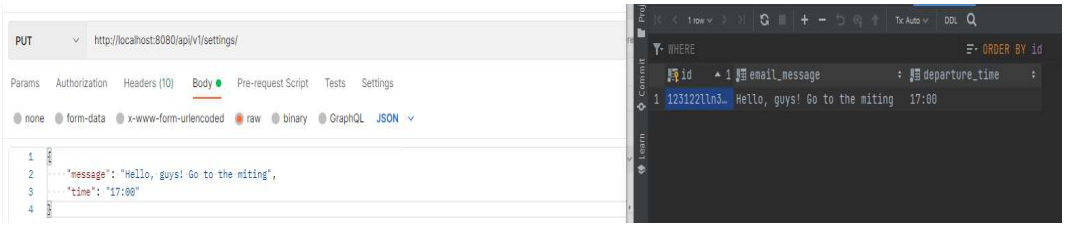


Рисунок 1.31 – Запрос на обновление сообщения и времени

На рисунках 1.32 и 1.33 представлена статистика, которую можно получить, как и за все время, так и за определенный год.

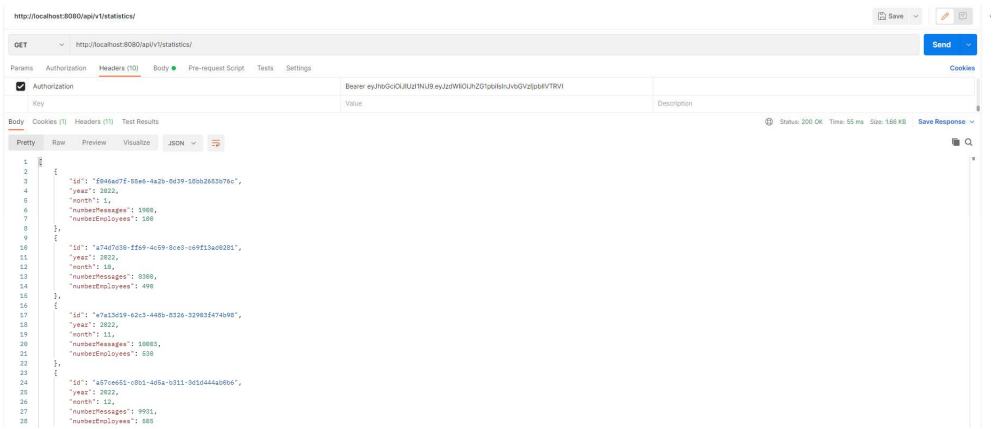


Рисунок 1.32 – Запрос на получение статистики за все время

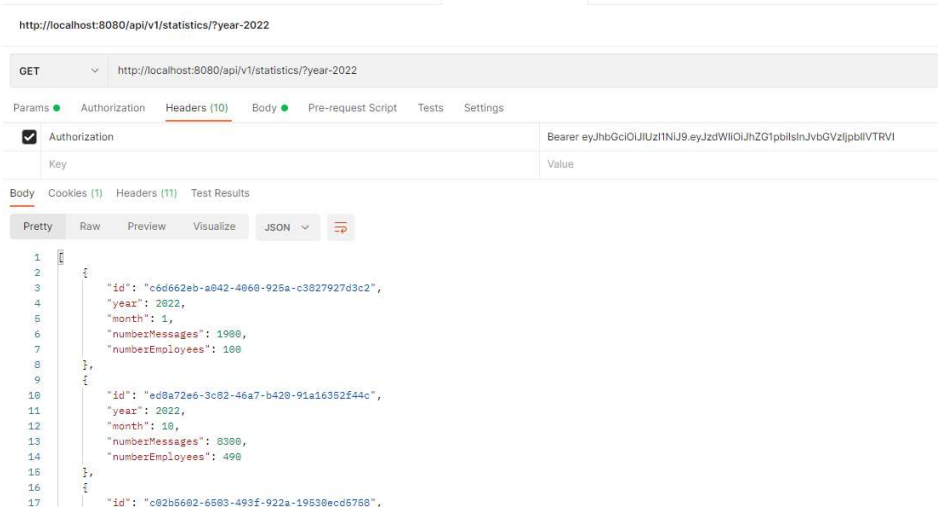


Рисунок 1.33 – Запрос на получение статистики за определенный год

Пример полученного сообщения (рисунок 1.34).

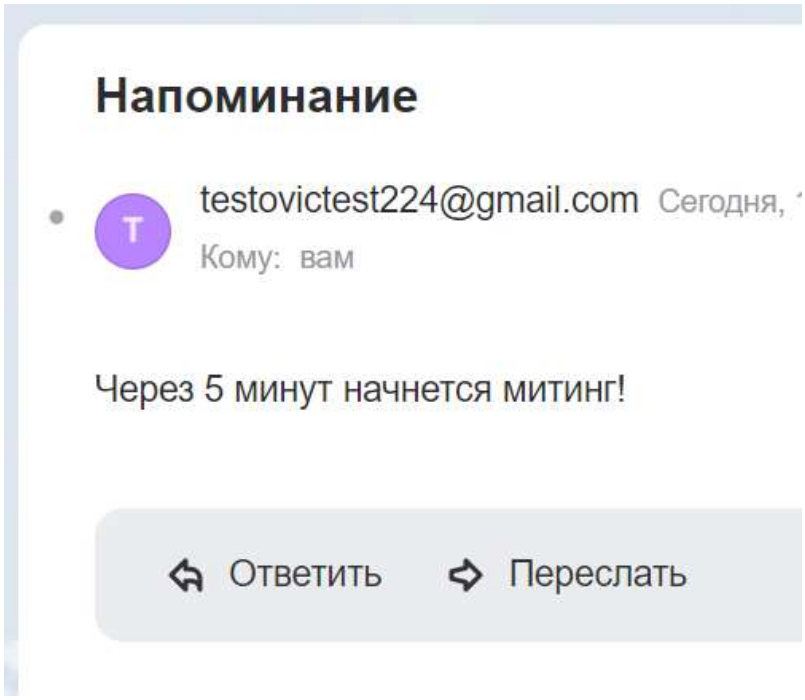


Рисунок 1.34 – Пример полученного сообщения

При нажатии на кнопку *«API»* появляется страница с доступным API, которую можно увидеть на рисунке 1.35.

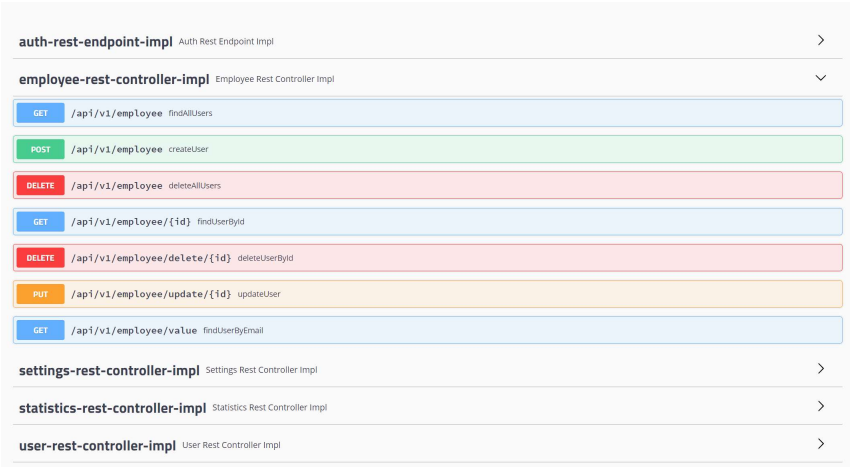


Рисунок 1.35 – Страница с доступным *API*

2 Охрана труда

2.1 Анализ негативных факторов, воздействующие на человека

Чрезмерное использование компьютера и интернета может привести к нескольким проблемам со здоровьем. Расширенное использование компьютера может влиять как на физиологические, так и на психические факторы здоровья. Это сопровождается плохой осанкой, скелетно-мышечными болями, ожирением, слабым зрением и психическими заболеваниями. Опасность для здоровья использования компьютера жизнь требует серьезного беспокойства.

Для уменьшения пагубного воздействия ПЭВМ на человека, необходимо соблюдать ряд правил использования персонального компьютера. Главными, из которых являются:

– правильная эргономика рабочего места;

– правила перерывов и режима работы;

– правила электробезопасности;

– освещенность рабочего места.

2.1.1 Эргономика рабочего места

Ниже представлены основные требования, которым должно удовлетворять рабочее место:

- необходим стул, который поддерживает изгибы позвоночника, где можно отрегулировать высоту так, чтобы ноги лежали на полу или на подставке для ног, а бедра были параллельны полу. Подлокотники должны находится в том положение, в котором руки мягко опираются на них, а плечи расслаблены;

- местоположение мыши должно находиться в легкой досягаемости и на той же поверхности, что и клавиатура. При наборе текста или

использовании мыши запястья нужно держать прямыми, предплечье близко к телу, а руки на уровне локтей или чуть ниже;

- под столом должен быть зазор для коленей, бедер и ног. Если стол слишком низкий и его нельзя отрегулировать. Если стол слишком высокий и его нельзя отрегулировать, то нужно поднять стул;

- монитор необходимо расположить на расстоянии вытянутой руки. Верхняя часть экрана должна быть на уровне глаз или чуть ниже. Монитор должен находиться непосредственно за клавиатурой. При ношении бифокальных очков, нужно опустить монитор еще на 1-2 дюйма для более удобного просмотра. Монитор располагается так, чтобы самый яркий источник света находился сбоку.

2.1.2 Правила перерывов и режима работы

Работник, работающий с ПК, обязан делать перерывы, чтобы быть более эффективным и не переутомляться. Для он должен:

- сократить длительное компьютерное время, когда это возможно;

- разбить работу на более мелкие сегменты и переключаться между задачами, которые используют разные движения. Например, чередовать использование мыши с чтением и поиском в Интернете;

- по крайней мере каждые 10 минут делать короткий (10-20 секунд) перерыв. Убирать руки с клавиатуры и двигаться;

- каждые 30-60 минут делать короткий (2-5 минут) перерыв, чтобы потянуться или прогуляться.

2.1.3 Электробезопасность

ПЭВМ подключен к электросети с напряжением 220 В, поэтому представляет для человека потенциальную опасность. Любое воздействие тока может привести к сильному ожогу и к нарушению работы внутренних органов.

Для предотвращения подобных ситуаций необходимо соблюдать требования безопасности, прописанные в ГОСТ 12.2.007-75. Основные меры предосторожности для соблюдения:

- розетки электропитания должны соответствовать вилкам электропитания;

- во время работы персонального компьютера запрещается подключать и отключать кабели соединения;

- все электрические розетки должны иметь маркировку по напряжению;

- значение номинального напряжения сети необходимо наносить яркой краской, крупными символами (высотой не менее 50 мм) на стене или щите, возле розетки.

В время компьютерной техники каждый сотрудник должен внимательно и ответственно относится к электропроводке, приборам и аппаратам, всегда помнить, что несоблюдение правил электробезопасности угрожает здоровью и жизни человека.

Необходимо постоянно следить на своем рабочем месте за исправным состоянием электропроводки, выключателей, штепсельных розеток, при помощи которых оборудование включается в сеть.

При обнаружении неисправностей необходимо обесточить электрооборудование. Продолжение работы возможно только после устранения всех неисправностей.

2.1.4 Освещенность рабочего места

Как естественное, так и искусственное (местное) освещение влияет на наше здоровье и благополучие. Это напрямую влияет на наше настроение и бдительность. Влияние освещения на зрение является наиболее очевидным воздействием на рабочую силу. Проблемы со зрением и дискомфорт возникают, когда освещение затрудняет работникам видеть то, что они должны видеть при выполнении своей работы, что, в свою очередь, может

негативно повлиять на их производительность. Аспекты освещения, которые обычно могут вызывать визуальный дискомфорт, - это недостаточное или чрезмерное освещение, тени, вуалирующие отражения, блики и мерцание. Исходя из этого, рассчитаем параметры искусственного освещения.

Расчет освещения производится для комнаты: длина (а) - 6 м, ширина (б) - 3 м, высота (*h*) – 3 м. Расчет производится под типовой растровый светильник с 4-ю линейными люминесцентными лампами (Кл), мощностью 18 Вт каждая (1 лампа даёт световой поток (СПл), равный 900 лк).

Поскольку должность программиста предполагает длительные монотонные операции — со высоким уровнем зрительной работы (различение объектов, размером от 3 до 5 мм), то возьмём за норму – освещённость его рабочего места (Е) в 500 люкс.

Коэффициент запаса обычно равен 1,2 (Кз).

Определяем индекс помещения (Ип), расчет:

$$Ип = S/((h1-h2) * (а+б)) = (6*3)/((3-0,8)*(6+3)) \approx 0,9.$$

Для данного индекса помещения коэффициент использования (U) примерно равен 65.

Выполним, расчет количества светильников (Ксв.) для данного помещения [7]:

$$Ксв. = (E*S*100*Кз)/(U*Кл*СПл).$$

Следовательно:

$$Ксв. = (500*18*100*1,2)/(65*4*900) = 5 \text{ светильников}$$

Получается, что для кабинета программиста размером 5 х 4 м достаточно будет установить 5 растровых светильников.

Заключение

В ходе выполнения выпускной квалификационной работы был разработан микросервис для автоматизации информирования сотрудников организаций.

На основании проделанной работы микросервис позволяет:

- автоматически отправлять сообщения о назначенных мероприятиях сотрудникам организации;
- сверять каждый день с официальным производственным календарем;
- управление всеми процессами с помощью удобного пользовательского интерфейса или *API*.

Во время выполнения данной работы были получены навыки разработки микросервиса на языке *Java* с использованием *Spring Boot* и системой управления базой данных *MySql*, а также реализации безопасного *REST API* с использованием *JWT*.

Программа написана с использование среды программирования *IntelliJ IDEA Ultimate Edition*.

В дальнейшем можно усовершенствовать разработанный микросервис и добавить *Refresh-token*, который умеет выполнять одну задачу – получение нового *JWT* для доступа к *API* микросервиса. У *Refresh-token* есть особенность - они долгоживущие, но одноразовые. Также можно добавить разделение сотрудников на команды, у которых будут свои собственные напоминающие планировщики. В добавок для улучшение существующей безопасности можно внедрить использование *https* протокола, который обеспечивает защиту канала передачи данных, накладывая дополнительные криптографические протоколы – *SSL* и *TLS*.

Список использованных источников

1. Ньюмен С. Создание микросервисов - СПб.: «Питер» 2016. - 246 с.
2. baeldung.com: [Электронный ресурс]. URL: https://www.baeldung.com/ (дата обращения:23.04.2022)
3. spring.io: [Электронный ресурс]. URL: https://spring.io/(дата обращения:27.04.2022)
4. java.com: [Электронный ресурс]. URL: https://www.java.com/ru/ (дата обращения:28.04.2022)
5. maven.apache.org: [Электронный ресурс]. URL: https://www.maven.apache.org (дата обращения:30.04.2022)
6. Стивенс Р. Алгоритмы. Теория и практическое применение // John Wiley & Sons.-2013. - 72 с.
7. 1posvetu.ru: [Электронный ресурс]. URL: https://1posvetu.ru/montazh-i-nastrojka/raschet-osveshheniya-rabochego-mesta.html (дата обращения: 4.05.2022)

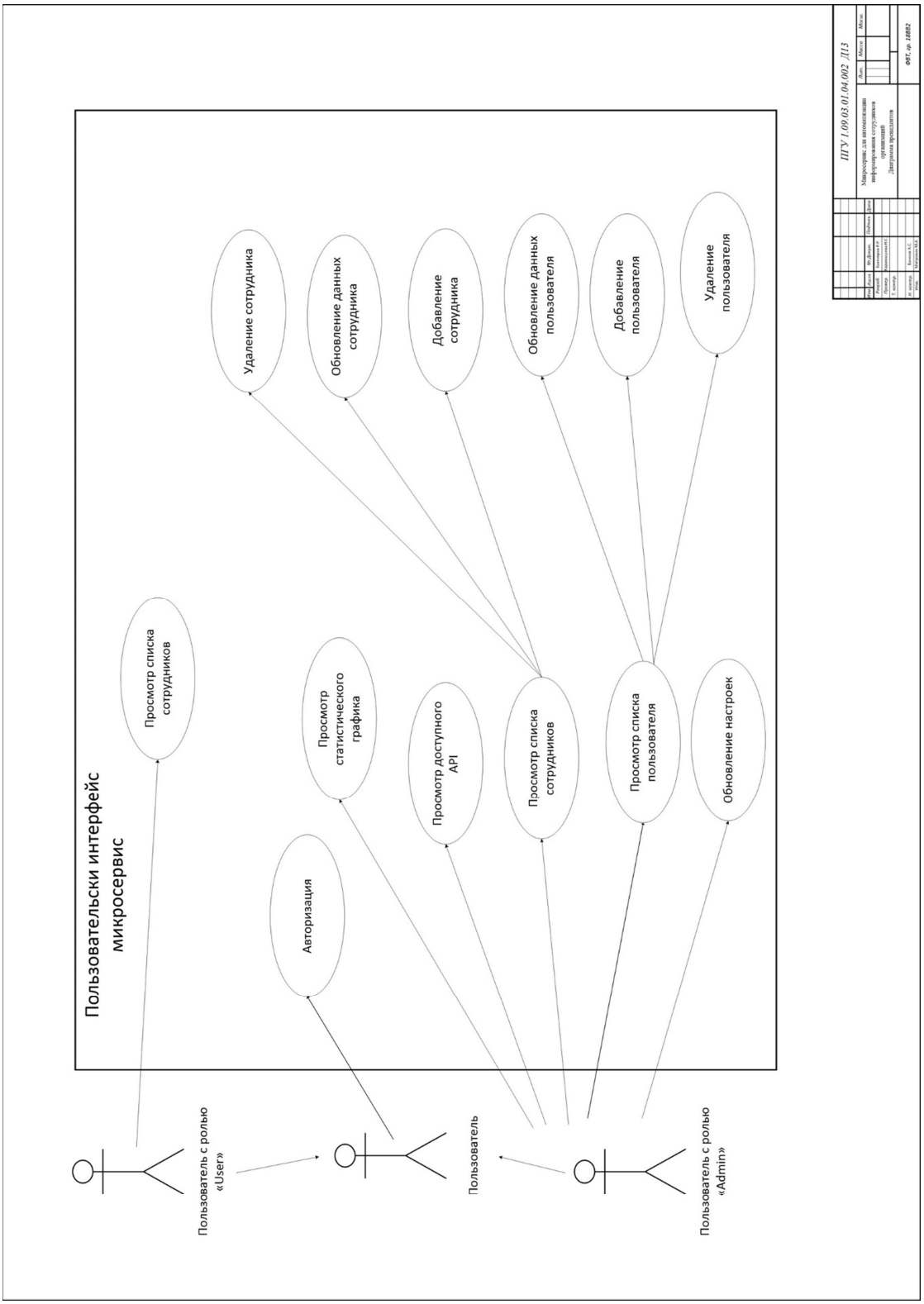
Приложение А

(Обязательное)

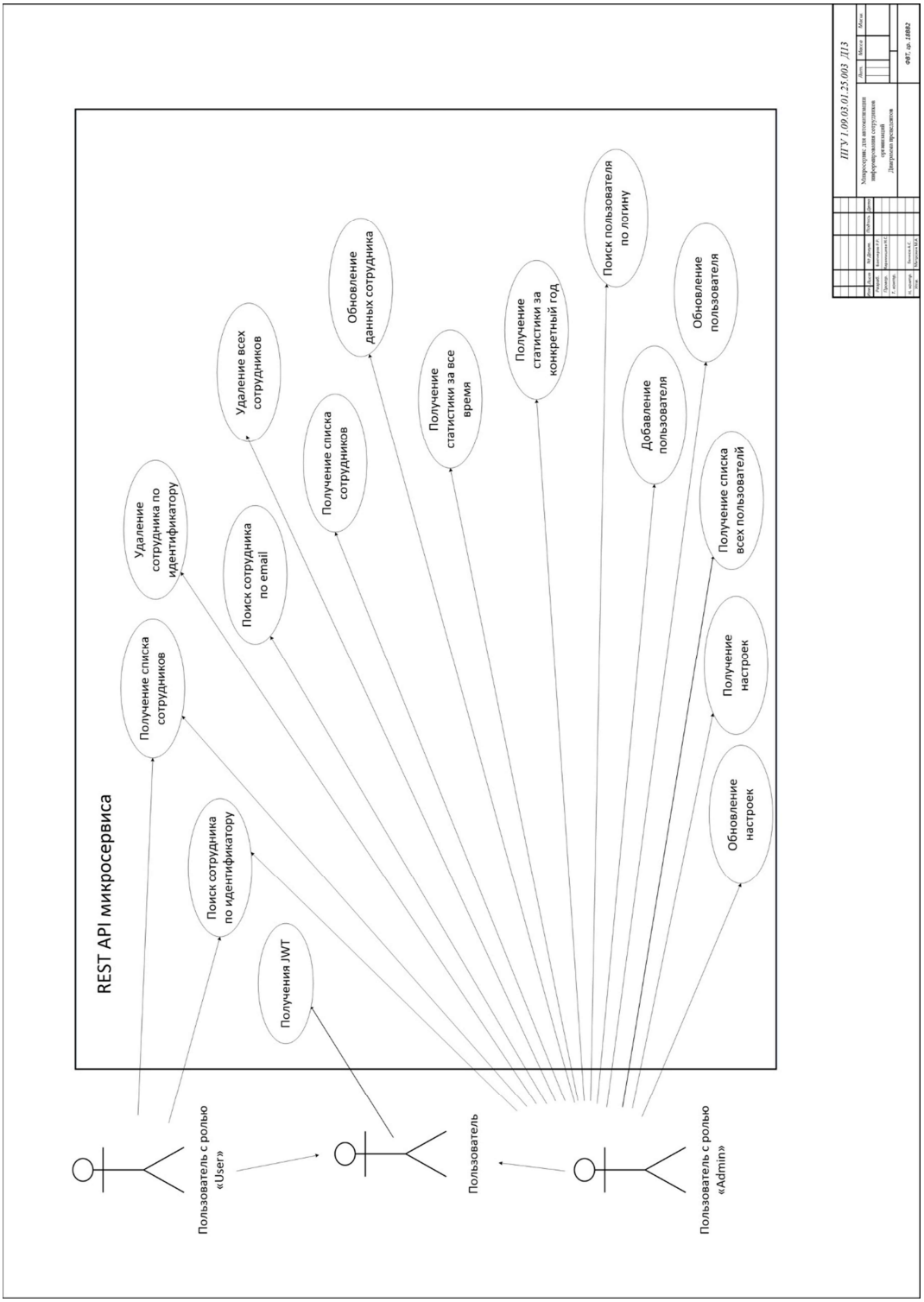
Микросервис для автоматизации информирования сотрудников организаций

UML - диаграммы

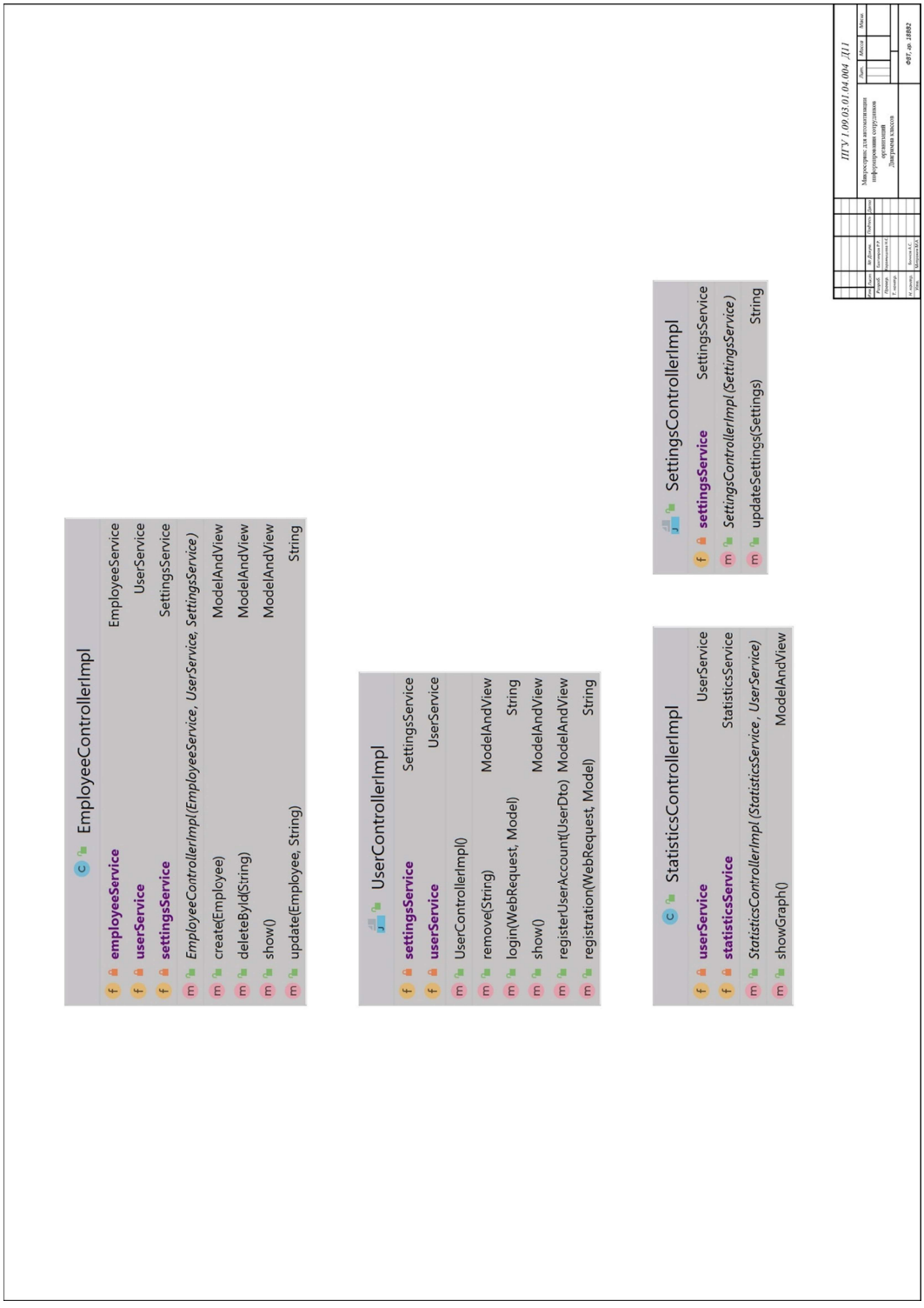
Приложение А.1 – UML – диаграмма прецедентов для пользовательского интерфейса



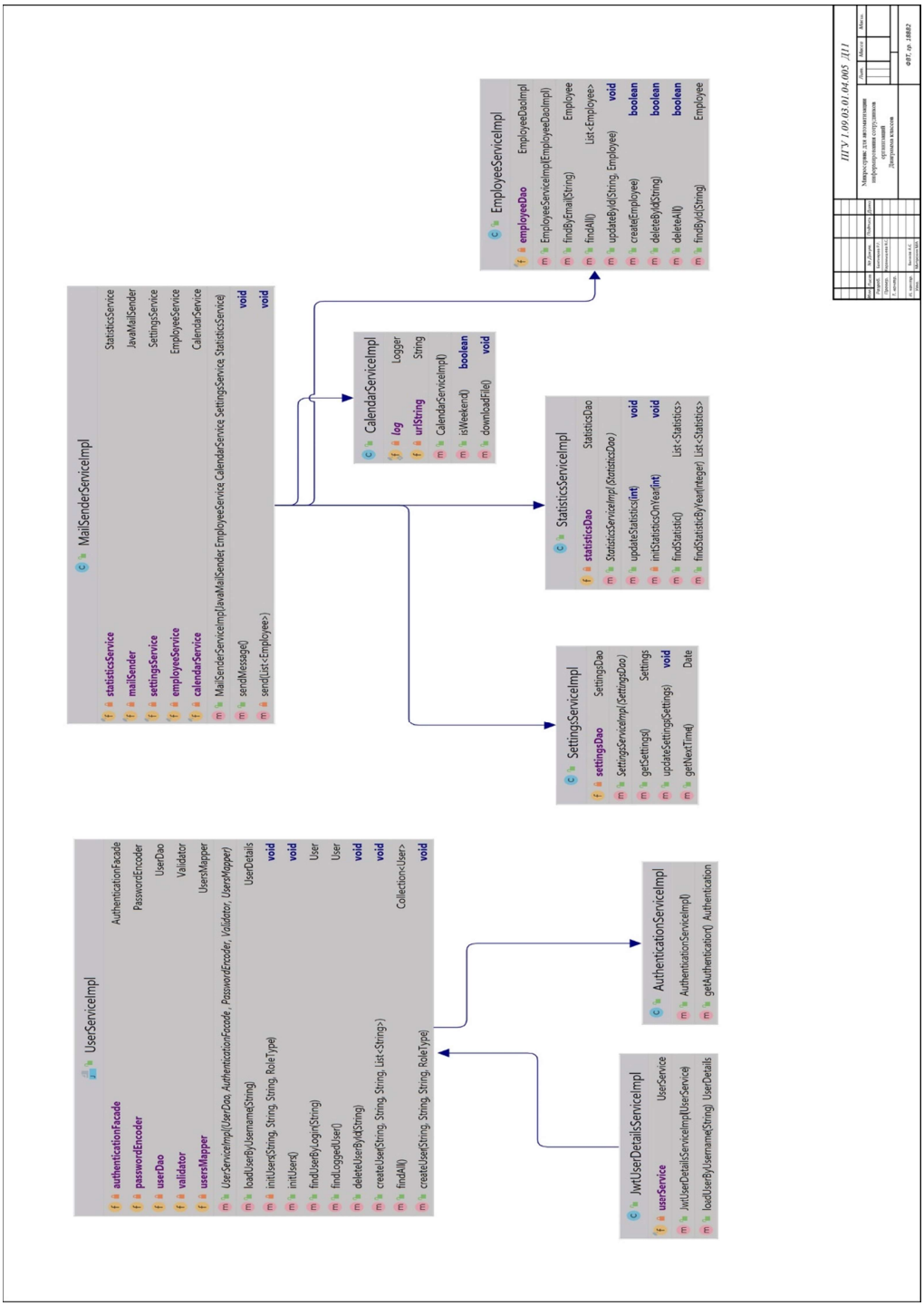
Приложение А.2 – UML – диаграмма прецедентов для *REST API*



Приложение А.3 – UML – диаграмма классов контроллеров



Приложение А.4 – UML – диаграмма классов сервисов



Приложение А.5 – UML – диаграмма классов слоя хранения данных

Приложение Б

(обязательное)

Микросервис для автоматизации информирования сотрудников организаций

Листинг программы

Файл «EmployeeDaoImpl.java»

```
package ru.bakhtiyarov.emailsender.dao.impl;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import ru.bakhtiyarov.emailsender.dao.EmployeeDao;
import ru.bakhtiyarov.emailsender.entity.Employee;
import ru.bakhtiyarov.emailsender.mapper.EmployeeMapper;

import java.util.List;

@Repository
public class EmployeeDaoImpl implements EmployeeDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public boolean create(@NotNull final Employee employee) {
        @NotNull final String SQL_INSERT_USER =
            "INSERT into employee (id, email, firstName, lastName)
values(?,?,?,?)";
        return jdbcTemplate.update(SQL_INSERT_USER, employee.getId(),
employee.getEmail(),
            employee.getFirstName(), employee.getLastName()) > 0;
    }

    @Override
    @NotNull
    public List<Employee> findAll() {
        @NotNull final String SQL_FIND_ALL = "SELECT * FROM employee";
        return jdbcTemplate.query(SQL_FIND_ALL, new EmployeeMapper());
    }

    @Override
    @Nullable
    public Employee findByEmail(final String email) {
        @NotNull final String SQL_FIND_ALL_BY_FIRST_NAME = "SELECT * FROM
employee where email = ?";
        return jdbcTemplate.queryForObject(SQL_FIND_ALL_BY_FIRST_NAME, new
EmployeeMapper(), email);
    }

    @Override
    @Nullable
    public Employee findById(@NotNull final String id) {
        @NotNull final String SQL_FIND_USER_BY_ID = "SELECT * FROM employee
where id = ?";
        return jdbcTemplate.queryForObject(SQL_FIND_USER_BY_ID, new
EmployeeMapper(), id);
    }

    @Override
    public boolean deleteById(@NotNull final String id) {
        @NotNull final String SQL_DELETE_USER_BY_ID = "DELETE FROM employee
where id = ?";
        @NotNull final Object[] args = new Object[]{id};
        return jdbcTemplate.update(SQL_DELETE_USER_BY_ID, args) > 0;
```

```
    }

    @Override
    public boolean deleteAll() {
        @NotNull final String SQL_DELETE_ALL_USERS = "DELETE FROM employee";
        return jdbcTemplate.update(SQL_DELETE_ALL_USERS) > 0;
    }

    @Override
    public void update(Employee employee) {
        @NotNull final String SQL_UPDATE_USER =
            "UPDATE employee SET id = ?, email = ?, firstName = ?, lastName =
? WHERE id = ?";
        jdbcTemplate.update(SQL_UPDATE_USER, employee.getId(),
employee.getEmail(),
            employee.getFirstName(), employee.getLastName(),
employee.getId());
    }
}
```

Файл «UserDaoImpl.java»

```
package ru.bakhtiyarov.emailsender.dao.impl;

import lombok.AllArgsConstructor;
import org.jetbrains.annotations.NotNull;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import ru.bakhtiyarov.emailsender.dao.UserDao;
import ru.bakhtiyarov.emailsender.entity.Role;
import ru.bakhtiyarov.emailsender.entity.User;
import ru.bakhtiyarov.emailsender.entity.UserDb;
import ru.bakhtiyarov.emailsender.mapper.UserMapper;

import java.util.List;

@Repository
@AllArgsConstructor
public class UserDaoImpl implements UserDao {

    final String SQL_INSERT_USER =
        "INSERT into app_user (id, login, email, password) values(?,?,?,?)";

    final String SQL_INSERT_ROLE =
        "INSERT into user_role (id, user_id, role_type) values(?,?,?)";

    final String SQL_SELECT_USER_BY_LOGIN =
        "SELECT  user.id, user.login, user.password, user.email,
role.role_type " +
            "FROM (SELECT id, login, password, email FROM app_user HAVING
login = ?) AS user " +
            "INNER JOIN user_role AS role ON user.id = role.user_id";

    final String SQL_SELECT_USERS =
        "SELECT  user.id, user.login, user.email, user.password,
role.role_type " +
            "FROM (SELECT id, login, email, password FROM app_user) AS user "
+
            "LEFT JOIN user_role AS role ON user.id = role.user_id";

    final String SQL_DELETE_USER = "DELETE FROM app_user where id = ?";
```

```
    private JdbcTemplate jdbcTemplate;

    @NotNull
    @Override
    public List<UserDb> findAll() {
        return jdbcTemplate.query(SQL_SELECT_USERS, (rs, rowNum) ->
            new UserDb()
                .setId(rs.getString("user.id"))
                .setLogin(rs.getString("user.login"))
                .setEmail(rs.getString("user.email"))
                .setRole(rs.getString("role.role_type")));
    }

    @Override
    @Transactional
    public void save(User user, Role role) {
        jdbcTemplate.update(SQL_INSERT_USER, user.getId(), user.getLogin(),
            user.getEmail(),
            user.getPassword());
        jdbcTemplate.update(SQL_INSERT_ROLE, role.getId(), user.getId(),
            role.getRoleType().name());
    }

    @Override
    @Transactional
    public void save(User user) {
        jdbcTemplate.update(SQL_INSERT_USER, user.getId(),  user.getLogin(),
user.getEmail(), user.getPassword());
        user.getRoles()
            .forEach(val ->
                jdbcTemplate.update(SQL_INSERT_ROLE, val.getId(),
user.getId(),
                    val.getRoleType().name()));
    }

    @Override
    public User findByLogin(String login) {
        try {
            return jdbcTemplate.queryForObject(SQL_SELECT_USER_BY_LOGIN, new
UserMapper(), login);
        } catch (Exception e) {
            return null;
        }
    }

    @Override
    public void deleteById(String id) {
        jdbcTemplate.update(SQL_DELETE_USER, id);
    }

}
```

Файл «StatisticsDaoImpl.java»

```
package ru.bakhtiyarov.emailsender.dao.impl;

import lombok.AllArgsConstructor;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import ru.bakhtiyarov.emailsender.dao.StatisticsDao;
import ru.bakhtiyarov.emailsender.entity.Statistics;
```

```
import java.util.List;

@Repository
@AllArgsConstructor
public class StatisticsDaoImpl implements StatisticsDao {

    private JdbcTemplate jdbcTemplate;

    final String SQL_UPDATE_STATISTICS =
        "UPDATE statistics SET number_messages = ?, number_employees = ?
WHERE statistics_year = ? AND months = ?";

    final String SQL_SELECT_STATISTICS_BY_YEAR =
        "SELECT * FROM statistics WHERE statistics_year = ?";

    final String SQL_INSERT_STATISTICS =
        "INSERT INTO statistics (id, statistics_year, months,
number_messages, number_employees) value(?,?,?,?,?)";

    final String SQL_SELECT_STATISTICS_BY_YEAR_AND_MONTHS =
        "SELECT * FROM statistics WHERE statistics_year = ? AND months = ?";

    @Override
    public void insertStatistics(Statistics statistics) {
        jdbcTemplate.update(SQL_INSERT_STATISTICS, statistics.getId(),
            statistics.getYear(), statistics.getMonth(),
            statistics.getNumberMessages(), statistics.getNumberEmployees());
    }

    @Override
    public List<Statistics> findStatisticsByYear(int year) {
        return jdbcTemplate.query(SQL_SELECT_STATISTICS_BY_YEAR, ((rs, i) ->
            new Statistics()
                .setYear(rs.getInt("statistics_year"))
                .setMonth(rs.getInt("months"))
                .setNumberEmployees(rs.getInt("number_employees"))
                .setNumberMessages(rs.getInt("number_messages"))
        ), year);
    }

    @Override
    public Statistics findStatisticsByYearAndMonths(int year, int months) {
        return jdbcTemplate.query(SQL_SELECT_STATISTICS_BY_YEAR_AND_MONTHS,
            rs -> rs.next() ? new Statistics()
                .setYear(rs.getInt("statistics_year"))
                .setMonth(rs.getInt("months"))
                .setNumberEmployees(rs.getInt("number_employees"))
                .setNumberMessages(rs.getInt("number_messages")) : null,
            year, months);
    }

    @Override
    public void updateStatistics(Statistics statistics) {
        jdbcTemplate.update(SQL_UPDATE_STATISTICS,
statistics.getNumberMessages(),
            statistics.getNumberEmployees(), statistics.getYear(),
statistics.getMonth());
    }

}
```

Файл «SettingsDaoImpl.java»

```
package ru.bakhtiyarov.emailsender.dao.impl;

import lombok.AllArgsConstructor;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import ru.bakhtiyarov.emailsender.dao.SettingsDao;
import ru.bakhtiyarov.emailsender.entity.Settings;

@Repository
@AllArgsConstructor
public class SettingsDaoImpl implements SettingsDao {

    private JdbcTemplate jdbcTemplate;

    @Override
    public void updateSettings(Settings settings) {
        final String SQL_UPDATE_SETTINGS = "UPDATE settings SET email_message
= ?, departure_time = ?";
        jdbcTemplate.update(SQL_UPDATE_SETTINGS, settings.getMessage(),
settings.getTime());
    }

    @Override
    public Settings getSettings() {
        final String SQL_FIND_SETTINGS = "SELECT * FROM settings";
        return jdbcTemplate.queryForObject(SQL_FIND_SETTINGS, (rs, rowNum) ->
            new Settings()
                .setMessage(rs.getString("email_message"))
                .setTime(rs.getString("departure_time")));
    }
}
```

Файл «CalendarServiceImpl.java»

```
package ru.bakhtiyarov.emailsender.service.impl;

import com.opencsv.CSVReader;
import com.opencsv.exceptions.CsvException;
import lombok.extern.log4j.Log4j2;
import org.apache.commons.io.FileUtils;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import ru.bakhtiyarov.emailsender.service.CalendarService;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.net.URL;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Arrays;

@Log4j2
@Service
public class CalendarServiceImpl implements CalendarService {

    @Value("${calendar.url}")
    private String urlString;
```

```
    @Override
    public void downloadFile() {
        try {
            LocalDateTime localDateTime = LocalDateTime.now();
            if (localDateTime.getMonth().getValue() == 1 &&
localDateTime.getDayOfYear() == 1) {
                final File productionCalendarFile = new File("calendar.csv");
                final URL url = new URL(urlString);
                FileUtils.copyURLToFile(url, productionCalendarFile);
            }
        } catch (IOException e) {
            log.warn(e.getMessage());
        }
    }

    @Override
    public boolean isWeekend() {
        final ZonedDateTime now = ZonedDateTime.now(ZoneId.of("GMT+3"));
        final int year = now.getYear();
        final int month = now.getMonth().getValue();
        final String day = String.valueOf(now.getDayOfMonth());
        final String weekEnds;

        final File calendarFile = new File("calendar.csv");
        try (final CSVReader reader = new CSVReader(new
FileReader(calendarFile))) {
            weekEnds = reader.readAll().stream()
                                       .skip(1)
                                       .filter(strings ->
strings[0].equals(String.valueOf(year)))
                                       .map(strings -> strings[month])
                                       .findFirst()
                                       .orElse(null);
        } catch (IOException | CsvException e) {
            log.warn(e.getMessage());
            return true;
        }
        if (StringUtils.isEmpty(weekEnds)) {
            log.warn("Error parsing file.");
            return true;
        }
        final String[] weekendsArray = weekEnds.split(",");
        return Arrays.stream(weekendsArray).anyMatch(s -> s.matches(day) ||
s.matches(day + '*'));
    }

}
```

Файл «EmployeeServiceImpl.java»

```
package ru.bakhtiyarov.emailsender.service.impl;

import org.apache.commons.lang3.StringUtils;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.bakhtiyarov.emailsender.dao.impl.EmployeeDaoImpl;
import ru.bakhtiyarov.emailsender.entity.Employee;
import ru.bakhtiyarov.emailsender.exception.*;
import ru.bakhtiyarov.emailsender.service.EmployeeService;

import java.util.List;
```

```
import java.util.UUID;

@Service
public class EmployeeServiceImpl implements EmployeeService {

    private final EmployeeDaoImpl employeeDao;

    public EmployeeServiceImpl(EmployeeDaoImpl employeeDao) {
        this.employeeDao = employeeDao;
    }

    @Override
    public boolean create( final Employee employee) {
        if (employee.getEmail() == null || employee.getEmail().isEmpty())
throw new EmptyEmailException();
        String id = UUID.randomUUID().toString();
        employee.setId(id);
        return employeeDao.create(employee);
    }


    @Override
    public List<Employee> findAll() {
        return employeeDao.findAll();
    }


    @Override
    public Employee findById(final String id) {
        if (id == null || id.isEmpty()) throw new UserNotFoundException();
        return employeeDao.findById(id);
    }


    @Override
    public Employee findByEmail(final String email) {
        if (StringUtils.isBlank(email)) throw new EmptyEmailException();
        return employeeDao.findByEmail(email);
    }

    @Override
    public boolean deleteById(final String id) {
        if (StringUtils.isBlank(id)) throw new EmptyIdException();
        return employeeDao.deleteById(id);
    }

    @Override
    public boolean deleteAll() {
        return employeeDao.deleteAll();
    }

    @Override
    public void updateById(String id, Employee employee) {
        if (StringUtils.isBlank(id)) throw new EmptyIdException();
        if (employee == null) throw new EmployeeNotFoundException();
        if (StringUtils.isBlank(employee.getFirstName())) throw new
EmptyFirstNameException();
        if (StringUtils.isBlank(employee.getLastName())) throw new
EmptyLastNameException();
        if (StringUtils.isBlank(employee.getEmail())) throw new
EmptyEmailException();
        final Employee employeeFroUpdate = employeeDao.findById(id);
```

```
        if (employeeFroUpdate == null) throw new EmployeeNotFoundException();
        employee.setFirstName(employee.getFirstName());
        employee.setLastName(employee.getLastName());
        employee.setEmail(employee.getEmail());
        employeeDao.update(employee);
    }

}
```

Файл «MailSenderServiceImpl.java»

```
package ru.bakhtiyarov.emailsender.service.impl;

import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;
import ru.bakhtiyarov.emailsender.constant.TeaOfferConst;
import ru.bakhtiyarov.emailsender.entity.Employee;
import ru.bakhtiyarov.emailsender.service.*;

import java.util.List;
import java.util.Objects;

@Service
public class MailSenderServiceImpl implements MailSenderService {

    private final JavaMailSender mailSender;
    private final EmployeeService employeeService;
    private final CalendarService calendarService;
    private final SettingsService settingsService;
    private final StatisticsService statisticsService;

    @Autowired
    public MailSenderServiceImpl(
        final JavaMailSender mailSender,
        final EmployeeService employeeService,
        final CalendarService calendarService,
        final SettingsService settingsService,
        final StatisticsService statisticsService
    ) {
        this.mailSender = mailSender;
        this.employeeService = employeeService;
        this.calendarService = calendarService;
        this.settingsService = settingsService;
        this.statisticsService = statisticsService;
    }

    @Override
    public void sendMessage() {
        calendarService.downloadFile();
        if (calendarService.isWeekend()) {
            return;
        }
        final List<Employee> employees = employeeService.findAll();
        send(employees);
    }

    private void send(final List<Employee> employees) {
        statisticsService.updateStatistics(employees.size());

        final SimpleMailMessage mailMessage = new SimpleMailMessage();
```

```
        String message = settingsService.getSettings().getMessage();

        mailMessage.setFrom(TeaOfferConst.SENDER_EMAIL);
        mailMessage.setSubject(TeaOfferConst.SUBJECT);
        mailMessage.setText(message);

        employees.stream()
            .filter(Objects::nonNull)
            .filter(user -> StringUtils.isBlank(user.getEmail()))
            .forEach(user -> {
                    mailMessage.setTo(user.getEmail());
                    mailSender.send(mailMessage);
                }
            );
    }

}
```

Файл «UserServiceImpl.java»

```
package ru.bakhtiyarov.emailsender.service.impl;

import lombok.AllArgsConstructor;
import org.apache.commons.lang3.StringUtils;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.springframework.context.annotation.Primary;
import
org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import ru.bakhtiyarov.emailsender.dao.UserDao;
import ru.bakhtiyarov.emailsender.entity.*;
import ru.bakhtiyarov.emailsender.exception.UserNotFoundException;
import ru.bakhtiyarov.emailsender.mapper.UsersMapper;
import ru.bakhtiyarov.emailsender.service.AuthenticationFacade;
import ru.bakhtiyarov.emailsender.service.UserService;
import ru.bakhtiyarov.emailsender.validator.Validator;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

@Service
@Primary
@AllArgsConstructor
public class UserServiceImpl implements UserDetailsService, UserService {

    private UserDao userDao;
    private AuthenticationFacade authenticationFacade;
    private PasswordEncoder passwordEncoder;
    private Validator validator;
    private UsersMapper usersMapper;

    @PostConstruct
```

```
    public void initUsers() {
        initUsers("test", "test", "test@ya.ru", RoleType.USER);
        initUsers("admin", "admin", "admin@ya.ru", RoleType.ADMIN);
    }


    public Collection<User> findAll() {
        List<UserDb> users = userDao.findAll();
        if (!users.isEmpty()) {
            return usersMapper.toUsers(users);
        }
        return Collections.emptyList();
    }

    @Override
    public User findUserByLogin(String login) {
        if (StringUtils.isBlank(login)) throw new UserNotFoundException();
        return userDao.findByLogin(login);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
         final User user = userDao.findByLogin(username);
        if (user == null) throw new UsernameNotFoundException(username);
        return user;
    }


    public User findLoggedUser() {
        final Authentication authentication =
authenticationFacade.getAuthentication();
        if (authentication instanceof AnonymousAuthenticationToken) throw new
UserNotFoundException();
        final String currentUserName = authentication.getName();
        return userDao.findByLogin(currentUserName);
    }

    public void createUser( final String login,  final String password,
final String email,  final RoleType roleType) {
        validator.userValidator(login, password);
        final String passwordHash = passwordEncoder.encode(password);
        final User user = new User();
        user.setLogin(login);
        user.setPassword(passwordHash);
        user.setEmail(email);
        final Role role = new Role();
        role.setRoleType(roleType);
        user.setRoles(Collections.singletonList(role));
        userDao.save(user, role);
    }

    public void createUser( final String login,  final String password,
final String email, List<String> roles) {
        validator.userValidator(login, password);
        final String passwordHash = passwordEncoder.encode(password);
        final User user = new User();
        user.setLogin(login);
        user.setPassword(passwordHash);
        user.setEmail(email);

        List<Role> roleTypes = roles.stream()
```

```
            .map(role -> new Role().setRoleType(RoleType.valueOf(role)))
            .collect(Collectors.toList());

        if (roleTypes.isEmpty()) {
            user.setRoles(Collections.singletonList(new
Role().setRoleType(RoleType.USER)));
        } else {
            user.setRoles(roleTypes);
        }
        userDao.save(user);
    }

    @Override
    public void deleteUserById(String id) {
        userDao.deleteById(id);
    }

    private void initUsers(final String login, String pass, String email,
RoleType roleType) {
        final User user = userDao.findByLogin(login);
        if (user != null) return;
        createUser(login, pass, email, roleType);
    }

}
```

Файл «EmployeeRestControllerImpl.java»

```
package ru.bakhtiyarov.emailsender.endpoint.impl;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import ru.bakhtiyarov.emailsender.endpoint.UserRestController;
import ru.bakhtiyarov.emailsender.entity.Employee;
import ru.bakhtiyarov.emailsender.service.EmployeeService;

import java.util.List;

@RestController
@RequestMapping("/api/v1/employee")
public class EmployeeRestControllerImpl implements UserRestController {

    @NotNull
    @Autowired
    private EmployeeService employeeService;

    @Override
    @PostMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public void createUser(
        @RequestHeader("Authorization") final String token,
        @NotNull @RequestBody final Employee employee
    ) {
        employeeService.create(employee);
    }

    @Override
    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public List<Employee> findAllUsers(
        @RequestHeader("Authorization") final String token
```

```
        ) {
        return employeeService.findAll();
    }

    @Override
    @NotNull
    @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public Employee findUserById(
        @RequestHeader("Authorization") final String token,
        @NotNull @PathVariable(name = "id") final String id
    ) {
        return employeeService.findById(id);
    }

    @Override
    @NotNull
    @GetMapping(value = "/value", produces =
MediaType.APPLICATION_JSON_VALUE)
    public Employee findUserByEmail(
        @RequestHeader("Authorization") final String token,
        @Nullable @RequestParam(value = "email", required = false) final
String email
    ) {
        return employeeService.findByEmail(email);
    }

    @PutMapping(value = "/update/{id}")
    public void updateUser(
        @RequestHeader("Authorization") final String token,
        @PathVariable("id") final @Nullable String id,
        @RequestBody final @Nullable Employee employee) {
        employeeService.updateById(id, employee);
    }


    @Override
    @DeleteMapping(value = "delete/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
    public void deleteUserById(
        @RequestHeader("Authorization") final String token,
        @NotNull @PathVariable(name = "id") final String id
    ) {
         employeeService.deleteById(id);
    }

    @Override
    @DeleteMapping()
    public void deleteAllUsers(
        @RequestHeader("Authorization") final String token
    ) {
        employeeService.deleteAll();
    }


}
```

Файл «UserRestControllerImpl.java»

```
package ru.bakhtiyarov.emailsender.endpoint.impl;

import lombok.AllArgsConstructor;
import org.jetbrains.annotations.NotNull;
```

```
import org.jetbrains.annotations.Nullable;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import ru.bakhtiyarov.emailsender.endpoint.UserController;
import ru.bakhtiyarov.emailsender.entity.User;
import ru.bakhtiyarov.emailsender.entity.UserDto;
import ru.bakhtiyarov.emailsender.mapper.UserMapper;
import ru.bakhtiyarov.emailsender.service.UserService;

import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

@RestController
@AllArgsConstructor
@RequestMapping("/api/v1/users/")
public class UserRestControllerImpl implements UserController {

    private UserService userService;
    private UserMapper userMapper;

    @Override
    @PostMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public void createUser(
        @NotNull @RequestBody final UserDto userDto
    ) {
        userService.createUser(userDto.getLogin(), userDto.getPassword(),
userDto.getEmail(), userDto.getRoles());
    }

    @Override
    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public List<UserDto> findAllUsers(
        @RequestHeader("Authorization") final String token
    ) {
        Collection<User> users = userService.findAll();
        return users.stream()
                    .map(user -> userMapper.toUserDto(user))
                    .collect(Collectors.toList());
    }

    @Override
    @NotNull
    @GetMapping(value = "/{login}", produces =
MediaType.APPLICATION_JSON_VALUE)
    public UserDto findUserByLogin(
        @Nullable @PathVariable(value = "login") final String login
    ) {
        return userMapper.toUserDto(userService.findUserByLogin(login));
    }

    @Override
    @DeleteMapping(value = "delete/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
    public void deleteUserById(
        @NotNull @PathVariable(name = "id") final String id
    ) {
        userService.deleteUserById(id);
    }

}
```

Файл «JwtTokenProvider.java»

```
package ru.bakhtiyarov.emailsender.security.jwt;

import io.jsonwebtoken.*;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;
import ru.bakhtiyarov.emailsender.exception.UserNotFoundException;

import javax.annotation.PostConstruct;
import javax.servlet.http.HttpServletRequest;
import java.util.Base64;
import java.util.Date;
import java.util.List;

@Component
public class JwtTokenProvider {

    @Value("${token.secret.key}")
    private String secret;

    @Value("${jwt.token.expired}")
    private long validityInMilliseconds;

    @Autowired
    @Qualifier("jwtUserDetailsServiceImpl")
    private UserDetailsService userDetailsService;

    @PostConstruct
    protected void init() {
        secret = Base64.getEncoder().encodeToString(secret.getBytes());
    }

    @NotNull
    public String createToken(@NotNull final String username, @NotNull final
List<String> roles) {
        @NotNull final Claims claims = Jwts.claims().setSubject(username);
        claims.put("roles", roles);
        @NotNull final Date now = new Date();
        @NotNull final Date validity = new Date(now.getTime() +
validityInMilliseconds);
        return Jwts.builder()
            .setClaims(claims)
            .setIssuedAt(now)
            .setExpiration(validity)
            .signWith(SignatureAlgorithm.HS256, secret)
            .compact();
    }

    public Authentication getAuthentication(@NotNull final String token) {
        @NotNull final UserDetails userDetails =
this.userDetailsService.loadUserByUsername(getUsername(token));
```

```
        return new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
    }

    @NotNull
    public String getUsername(@NotNull final String token) {
        return
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubjec
t();
    }

    @Nullable
    public String resolveToken(HttpServletRequest req) {
        @Nullable final String bearerToken = req.getHeader("Authorization");
        if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }

    public boolean validateToken(@NotNull final String token) {
        try {
            final Jws<Claims> claims =
Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return !claims.getBody().getExpiration().before(new Date());
        } catch (JwtException | IllegalArgumentException e) {
            throw new UserNotFoundException();
        }
```

Приложение В

(Обязательное)

Микросервис для автоматизации информирования сотрудников организаций

Презентация

МИКРОСЕРВИС ДЛЯ АВТОМАТИЗАЦИИ ИНФОРМИРОВАНИЯ СОТРУДНИКОВ ОРГАНИЗАЦИЙ

Автор работы:
Студент гр. 18ВВ2 Бахтияров Руслан Рафикович
Научный руководитель:
к.т.н., доцент каф. ВТ ПГУ Карамышева Н.С

Цель работы и объекты

Цель работы – разработать микросервис для автоматизации информирования сотрудников организаций о назначенных мероприятиях.

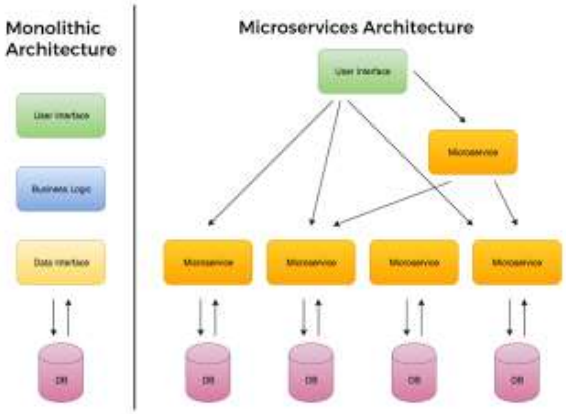
Объекты работы – микросервис, который позволяет:

- автоматически отправлять сообщения о назначенных мероприятиях сотрудникам организации;
- сверять каждый день с официальным производственным календарем;
- управлять всеми процессами с помощью удобного пользовательского интерфейса или API;
- безопасно авторизоваться.

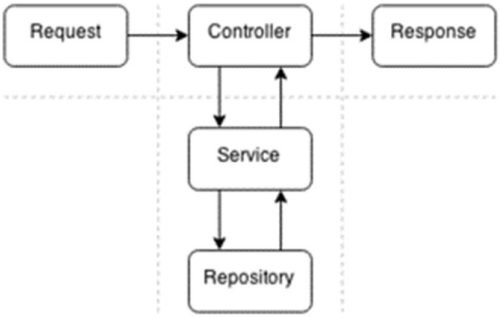
Популярные решения

Яндекс Почта

Архитектура

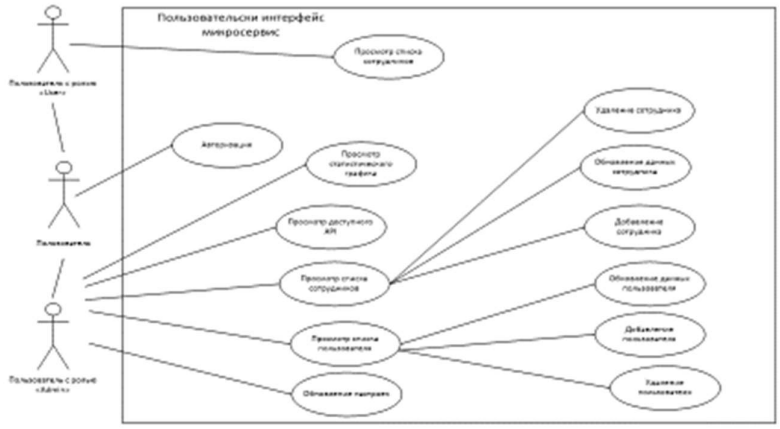


CONTROLLER-SERVICE-REPOSITORY



Стек технологий

UML – диаграмма прецедентов для UI



UML – диаграмма прецедентов для REST API

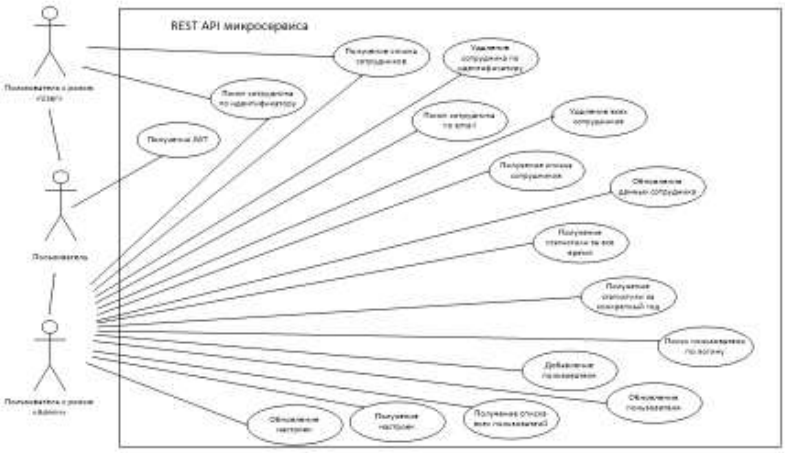


Диаграмма классов контроллеров

Диаграмма классов сервисов

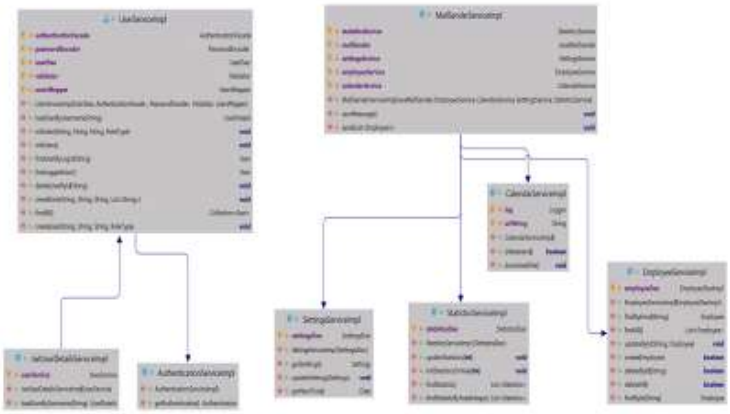
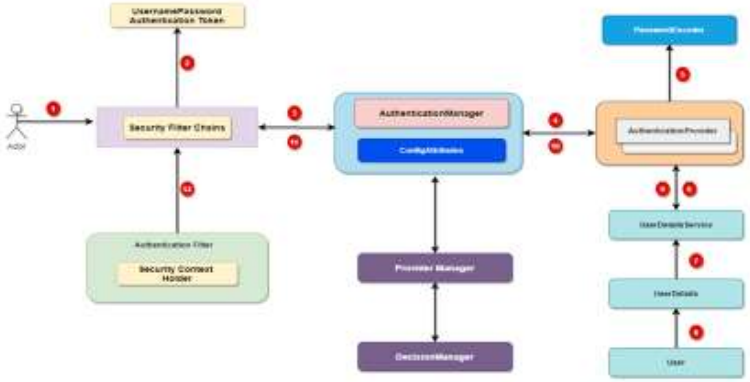
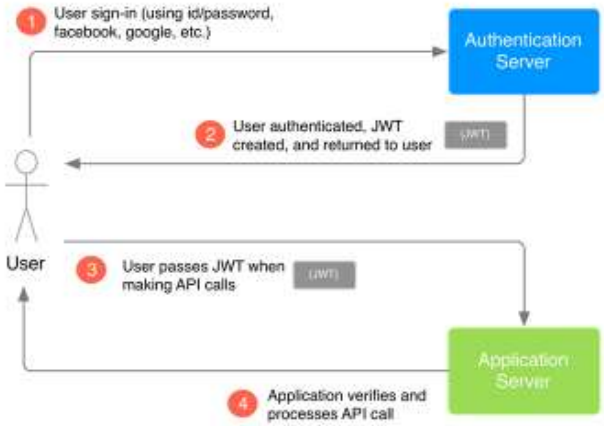


Диаграмма классов репозиториев

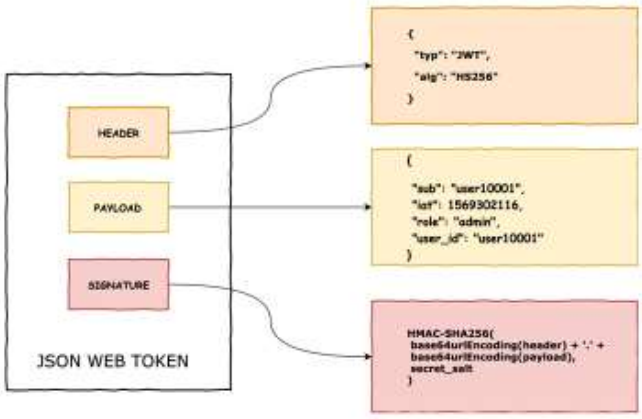
Безопасность Spring Security



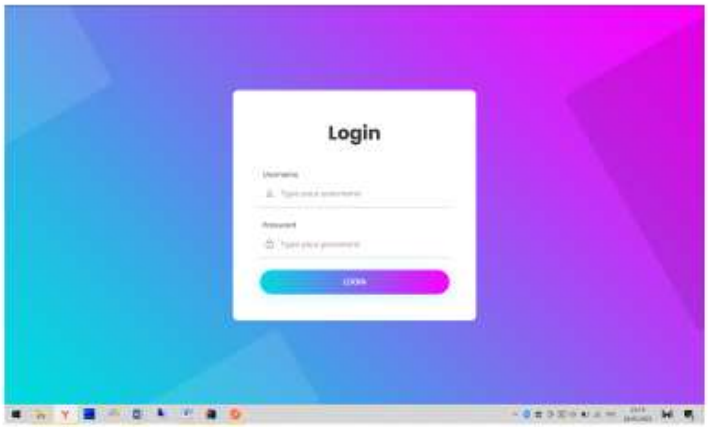
Безопасность JWT



Безопасность JWT

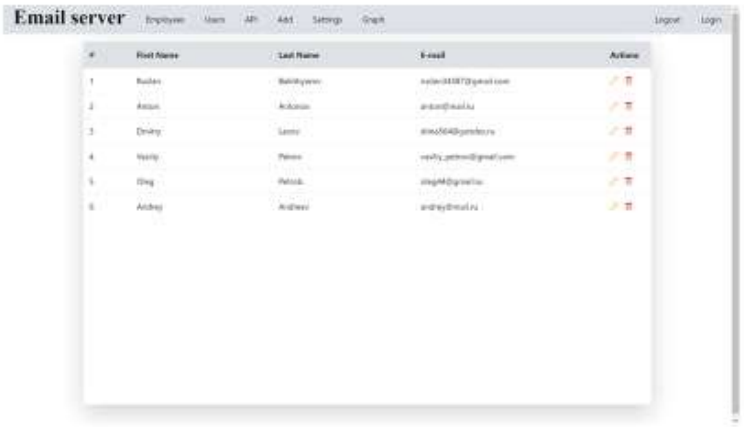


Форма авторизации

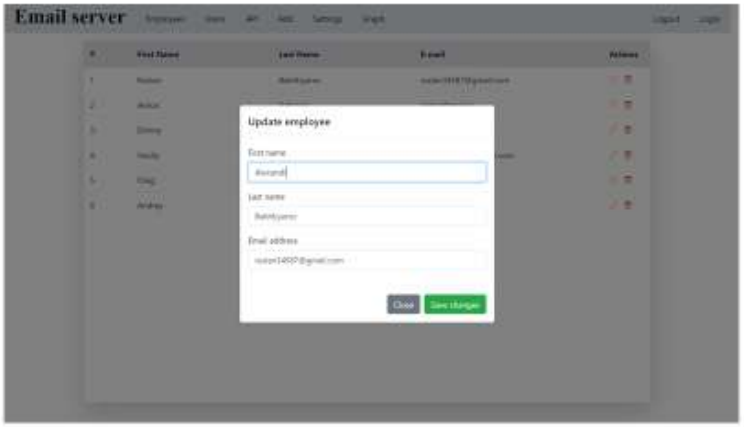


Главная страница пользователя с ролью «User»

Главная страница пользователя с ролью «Admin»
Таблица сотрудников



Форма редактирования данных сотрудника



Форма добавления сотрудника

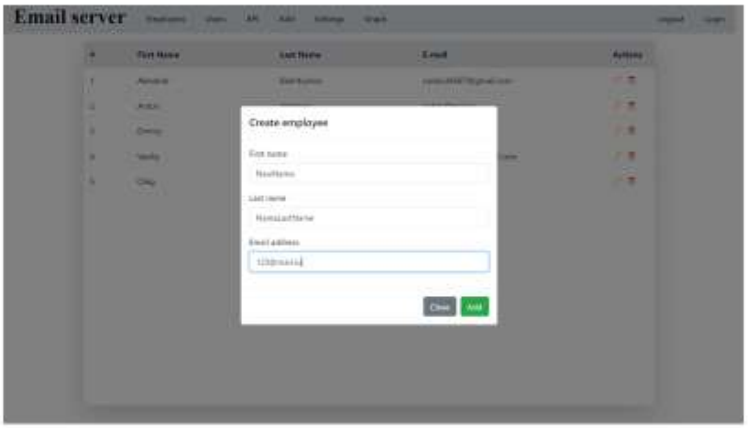
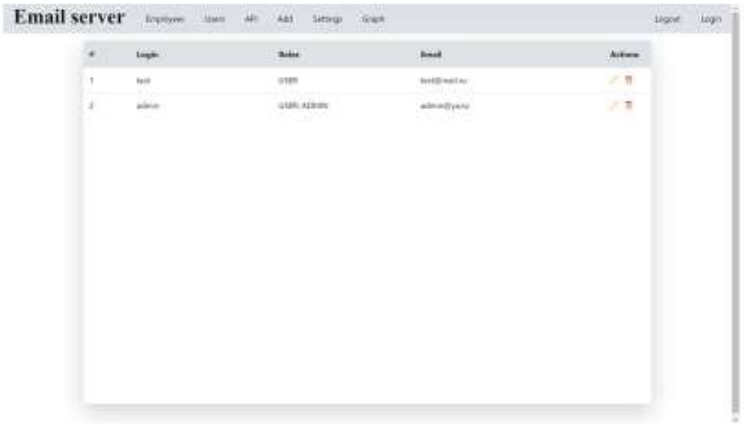
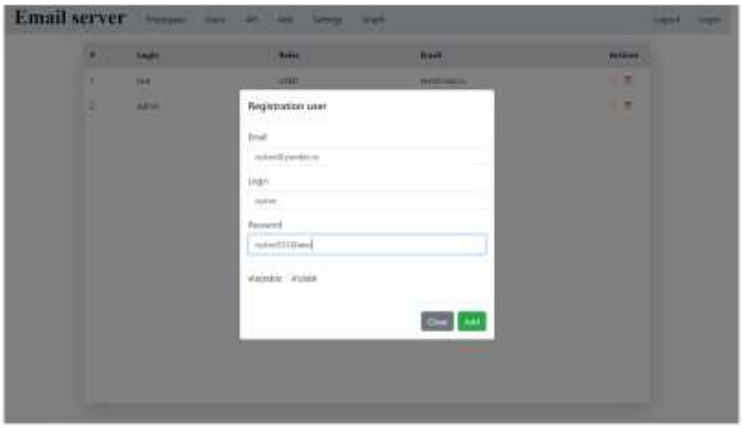


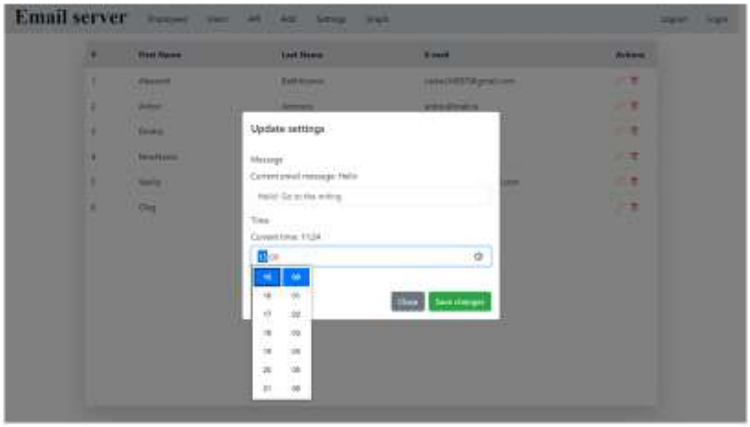
Таблица пользователей



Форма добавления нового пользователя



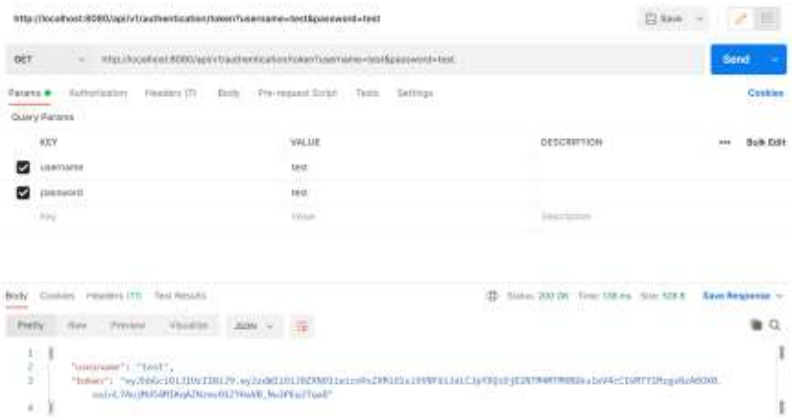
Форма обновления настроек



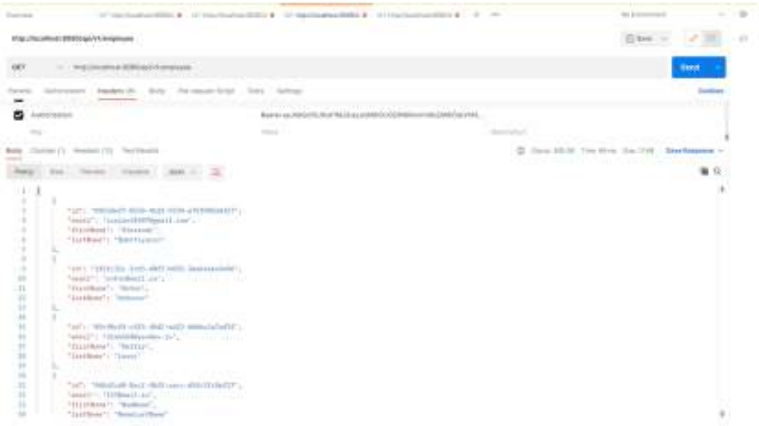
Статистический график

График, отображающий количество сотрудников

Запрос на получение *JWT*



Получение списка сотрудников организации



Поиск пользователя по логину

Результаты

На основании проделанной работы микросервис позволяет:

- автоматически отправлять сообщения о назначенных мероприятиях сотрудникам организации;

- сверять каждый день с официальным производственным календарем;

- управление всеми процессами с помощью удобного пользовательского интерфейса или API с использование JWT.

Благодарю за внимание!