МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Серверная часть веб-приложения «Почтовый клиент»

Студент: Белослудцев Егор Денисович
Группа: ИКБО-16-21
Работа представлена к защите 08.12.23 (дата) _____ / Белослудцев Е.Д. /
(подпись и ф.и.о. студента)

Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Работа допущена к защите 25/12 23 (дата) _____ / Синицын А.В./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: отл.
_____ / 25.12.23, Болбаков Р.Г., доцент /
_____ / 25.12.2023 Синицын А.В., старший преподаватель/
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка серверных частей интернет-ресурсов

Студент: Белослудцев Егор Денисович
Группа: ИКБО-16-21
Срок представления к защите: 08.12.2023
Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Тема: Серверная часть веб-приложения «Почтовый клиент»

Исходные данные: используемые технологии: HTML5, CSS3, Java, Jetbrains Intellij IDEA, SQL, наличие: межстраничной навигации, внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование паттерна проектирования (MVC, Clear Architecture, DDD), нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18, ГОСТ 7.32-2017.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Разработать слой клиентского представления веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «14» сентября 2023 г.

Задание на КР выдал: _____/А.В. Синицын/, «14» сентября 2023 г.

Задание на КР получил: _____/Е.Д. Белослудцев/, «14» сентября 2023 г.

АННОТАЦИИ

Руководитель курсовой работы: старший преподаватель А.В. Синицын.

Белослудцев Е.Д., Курсовая работа направления подготовки «Программная инженерия» на тему «Почтовый клиент»: М. 2023 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 47 стр., 17 рис., 20 источн. (в т.ч. 1 на английском яз.).

Ключевые слова: RESTful-приложение, MVC, проектирование, БД, Spring Boot, Spring Security, JWT, React, Thymeleaf, JSON.

Целью работы является проектирование веб-приложения для почтового клиента. Спроектирована информационная система, разработано решение для клиентской стороны, разработана база данных.

Belosludtsev E.D., Course work in the direction of training “Software Engineering” on the topic “Mail client”: M. 2023, MIREA - Russian Technological University (RTU MIREA), Institute of Information Technologies (IIT), Department of Tools and Application Software (IiPPO) – 47 p., 17 ill., 20 ref. (inc. 1 in English).

Keywords: RESTful application, MVC, design, BD, Spring Boot, Spring Security, JWT, React, Thymeleaf, JSON.

The purpose of the work is to design a web-application for a mail client. An information system was designed, a client-side solution was developed, a database was developed and a remote server was configured.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ_РСЧИР_ИКБО-16-21_БелослудцевЕД.pdf», исполнитель Белослудцев Е.Д.

ГЛОСАРИИ

1. **Почтовый клиент –** программное обеспечение, предназначенное для работы с электронной почтой. В контексте данной работы, это может включать в себя клиентскую часть, взаимодействующую с серверной частью.

2. **HTTP –** Протокол передачи гипертекста, используемый для обмена данными между клиентом и сервером.

3. **Аутентификация –** процесс проверки подлинности пользователя.

4. **Авторизация –** процесс предоставления доступа к определенным ресурсам или функционалу

5. **Серверная часть** – часть приложения, которая занимается обработкой сложных данных, взаимодействует с базой данных и может лишь передавать клиенту данные в специальном установленном формате.

6. **Клиентская часть –** часть приложения, отвечающая за представление данных, полученных от сервера, пользователю. Также клиентская часть, иначе – «визуальная часть», позволяет взаимодействовать с пользователем, принимая от него какую-то информацию и передавая её на сервер.

7. **Архитектура приложения** – правила разработки приложения, которые часто позволяют упростить разработку, поддержку и расширение приложения.

8. **База данных** – хранилище с данными, которые разработчик предпочел хранить отдельно от клиентской части (пользователи, сотрудники и т.п.). Такое хранилище управляется посредством серверной части и СУБД.

9. **СУБД (система управления базами данных)** – система, которая позволяет взаимодействовать с базой данных: добавлять, изменять, получать и удалять данные.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время электронная почта является неотъемлемой частью нашей повседневной жизни, играя ключевую роль в обмене информацией как на уровне личного общения, так и в бизнес-процессах. С развитием технологий и увеличением числа пользователей электронной почты возникает потребность в эффективных и надежных решениях для работы с данным видом коммуникации. В этом контексте, целью настоящей курсовой работы является разработка серверных частей интернет-ресурса "Почтовый клиент".

Современные почтовые клиенты часто ограничиваются простой клиентской частью, ориентированной на визуальный комфорт пользователя. Однако, при расширении функционала или бизнес-процессов, такие решения могут стать неэффективными. Разработка полноценного сервера для работы почтового клиента становится актуальной задачей, позволяя более гибко управлять данными, обеспечивать безопасность, и эффективно взаимодействовать с базой данных.

Объектом исследования данной курсовой работы является серверная часть интернет-ресурса "Почтовый клиент". Предметом исследования является разработка и анализ серверных компонентов этого ресурса, включая взаимодействие с базой данных, обеспечение безопасности и работу с почтовыми протоколами.

Основной целью данной работы является создание самодостаточного сервера, специально адаптированного для почтового клиента. Предполагается, что разработанный сервер будет работать в тесном взаимодействии с базой данных на платформе MySQL и будет реализован в соответствии с методологией REST и архитектурой MVC [1]. Это обеспечит гибкость в интеграции с различными клиентскими интерфейсами в будущем. Для достижения поставленной цели были поставлены следующие задачи:

1) сделать анализ предметной области,

2) выбрать технологии для разработки,

3) подобрать архитектуру веб – приложения,

4) разработать базу данных,

5) разработать серверную часть приложения,

6) разработать клиентскую часть приложения.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для установления необходимого функционала сервера были проанализированы 3 сайта с тематикой «почтовый клиент»:

- Gmail — популярный почтовый клиент, предоставляющий обширные возможности для электронной почты и интеграции с другими сервисами Google [2];
- Outlook — почтовый клиент от Microsoft с широким спектром функций, включая календарь, задачи и контакты, а также интеграцию с офисными приложениями [3];
- eM Client — клиент для управления электронной почтой с поддержкой календаря и задач, ориентированный на частных пользователей и бизнес [4].

1.1 Выделение ключевых аспектов

1.1.1 Интерфейс и удобство использования

Первый источник имеет современный и интуитивно понятный интерфейс с четко выделенными разделами и элементами управления. Обеспечивает простую навигацию, быстрый доступ к основным функциям, и наличие настраиваемых параметров для персонализации интерфейса.

Второй источник предлагает стильный и минималистичный дизайн с акцентом на визуальной привлекательности. Обеспечивает интуитивные жесты, анимации для повышения визуального опыта и возможности кастомизации интерфейса.

Последний веб – сервер обладает унифицированным интерфейсом, ориентированным на эффективное использование на различных устройствах. Предоставляет простые и понятные инструменты для управления почтовыми сообщениями, что способствует легкости в обучении.

Однако, стоит отметить, что у второго источника выявлены некоторые сложности в веб-серверной части, что может привести к временным задержкам при обработке запросов и отображении данных.

На основе проведенного анализа конкурентов можно сделать вывод о крайней важности обеспечения удобного и интуитивно понятного интерфейса

для пользователей. Первый и третий сайты продемонстрировали, что их пользовательский интерфейс спроектирован с учетом потребностей пользователей, обеспечивая легкость в использовании и высокую функциональность. Учитывая позитивный опыт данных проектов, следует уделить внимание созданию удобного интерфейса и в рамках разрабатываемого почтового клиента, для максимального удовлетворения потребностей конечных пользователей.

1.1.2 Функциональность

Первый источник предоставляет возможность добавления и управления несколькими учетными записями с различными настройками безопасности. Обеспечивает удобный интерфейс для просмотра, фильтрации и организации почтовых сообщений с возможностью быстрого поиска.

Второй источник и третьей источник обеспечивают интеграцию с аккаунтами на основных почтовых платформах и позволяет управлять ими в централизованном режиме. Предоставляет возможность группировки и сортировки писем, а также позволяет применять фильтры для автоматизации обработки. Имеет интегрированные средства антивирусной защиты и механизмы аутентификации для предотвращения несанкционированного доступа.

Исходя из проведенного анализа трех конкурирующих почтовых клиентов, делается вывод о ключевых аспектах, которые следует учесть при разработке собственного почтового приложения, а именно обеспечить пользователям возможность управления несколькими учетными записями. Также разрабатываемым приложение должно включать в себя реализацию функционала для удобного просмотра, фильтрации и организации почтовых сообщений, включая возможности быстрого поиска и автоматизации обработки.

1.1.3 Авторизация и аутентификация

Первый сайт обеспечивает разнообразные методы аутентификации, включая пароли, PIN-коды и биометрическую идентификацию, что повышает

уровень безопасности и удобства для пользователей.

Второй источник внедряет стандартные методы шифрования для защиты данных пользователя в процессе авторизации и взаимодействия с почтовым сервером.

Последний сайт реализует двухфакторную аутентификацию, дополнительно подтверждая личность пользователя для защиты от несанкционированного доступа.

При разработке механизма авторизации в почтовом приложении необходимо уделять особое внимание гибким методам аутентификации, обеспечению безопасности и шифрования данных. Это позволит создать максимально безопасное и удобное средство авторизации для пользователей.

1.1.4 Управление и организация

На приведенных выше источниках осуществлена Реализация механизмов фильтрации, сортировки и группировки сообщений помогает пользователям эффективно организовывать свою почту. Предоставление возможности быстрого поиска и автоматизации обработки сообщений также способствует более удобному управлению данными.

Также на втором сайте можно выделить внедрение системы меток и категорий позволяет пользователям более гибко организовывать и классифицировать почтовые сообщения, что улучшает общую структуру почтового ящика.

В контексте разрабатываемого почтового клиента управление и организация почтовых данных должны ориентироваться на обеспечение простоты, удобства и эффективности в работе пользователя. Сочетание персонализированных настроек и систем фильтрации позволит создать средство, легко адаптирующееся под индивидуальные потребности пользователей.

2 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

2.1 Выбор основных технологий

На основе проведенного анализа предметной области и рассмотрения конкурирующих почтовых клиентов, было принято решение выделить следующие ключевые технологии для реализации разрабатываемого почтового приложения: Spring, PostgreSQL и React.

2.1.1 Spring

Spring Framework является мощным инструментом для разработки серверных приложений, предоставляя обширный набор технологий и инструментов [5]. Одним из ключевых компонентов выбранного стека технологий является Spring Boot, который обеспечивает создание автономных приложений с минимальной конфигурацией [6]. Это особенно ценно для почтового клиента, где необходима высокая производительность и быстрый старт проекта.

Среди модулей Spring Framework, Spring Data JPA используется для удобного взаимодействия с базой данных через Java Persistence API (JPA) [7]. Это упрощает работу с данными и обеспечивает абстракцию от деталей реализации базы данных.

Spring Security представляет собой мощный инструмент для обеспечения безопасности приложений, включая аутентификацию, авторизацию и защиту от различных видов атак. В контексте почтового клиента, где безопасность является критическим аспектом, использование Spring Security становится важным шагом [8].

Одним из ключевых моментов в безопасности веб-приложений является обеспечение аутентификации пользователей. Spring Security предоставляет гибкие средства для настройки различных механизмов аутентификации, таких как формы входа, базовая аутентификация, аутентификация с использованием социальных сетей и другие [9].

Для обеспечения безопасности токенов аутентификации в почтовом клиенте, широко используется JWT (JSON Web Token) [10]. JWT представляет

собой стандартизированный формат для передачи информации между сторонами в виде JSON-объекта [11]. Он может быть подписан и/или зашифрован, обеспечивая безопасность передаваемых данных.

Spring Security упрощает внедрение JWT в аутентификацию. При успешной аутентификации пользователя, ему генерируется JWT токен, который включает необходимую информацию о пользователе и правах доступа. После этого токен передается клиенту, который должен включать его в каждый последующий запрос к защищенным ресурсам. Spring Security в свою очередь осуществляет проверку и обработку этого токена.

Использование JWT в Spring Security обеспечивает прозрачную и безопасную передачу данных между клиентом и сервером. Кроме того, это подход позволяет создавать расширенные системы авторизации, такие как управление сроком действия токена, обновление токена и многое другое.

Архитектура веб-приложения построена с применением Spring MVC. Ее модель-представление-контроллер (MVC) позволяет эффективно организовать код, разделяя бизнес-логику, представление и обработку запросов.

Для создания RESTful веб-сервисов используется модуль Spring RESTful Web Services, обеспечивая эффективное взаимодействие между клиентской и серверной частями приложения [12].

Проверка качества кода обеспечивается с использованием инструмента тестирования Spring Test, позволяя проводить как юнит-тестирование, так и интеграционное тестирование для обеспечения надежности и стабильности приложения. Выбор Spring и его модулей обусловлен стремлением к созданию высокопроизводительного, гибкого и безопасного почтового приложения [13].

2.1.2 PostgreSQL

PostgreSQL является мощной реляционной базой данных, обеспечивающей надежность, стабильность и поддержку сложных запросов. Ее открытый исходный код и активное сообщество разработчиков делают ее привлекательным выбором для почтового приложения. Способность

обеспечивать целостность данных и поддерживать транзакции делает PostgreSQL подходящим решением для хранения и управления информацией о пользователях и сообщениях [14].

2.1.3 React

В качестве фронтенд-технологии для разработки интерфейса почтового клиента рассматривалась возможность использования React [15]. Однако в контексте упрощения процесса рендеринга серверных страниц, также была рассмотрена альтернатива в виде Thymeleaf.

Thymeleaf предоставляет простой и удобный способ интеграции серверного и клиентского кода. Он позволяет создавать динамические HTML-страницы на сервере, внедряя данные напрямую в шаблоны. Это может быть полезным для упрощения процесса разработки и сокращения времени на создание интерфейса.

Однако, при использовании технологии JWT-токенов, возникают ограничения в применении динамических шаблонизаторов, таких как Thymeleaf. JWT-токены являются статичными, и предполагают передачу данных между клиентом и сервером в виде JSON-структур, что не совместимо с принципами динамического рендеринга Thymeleaf.

Таким образом, в выборе технологии для фронтенда учитывалась не только удобность разработки, но и требования по безопасной передаче данных между клиентом и сервером при использовании JWT-токенов. В итоге, решено было использовать более гибкую и современную технологию – React, для обеспечения более высокой производительности и масштабируемости почтового клиента.

2.1.4 Прочие технологии

При разработке почтового клиента, использование Java в качестве языка программирования является стратегическим выбором. Java предоставляет высокую универсальность, надежность и широкое применение в корпоративной среде. Этот язык программирования обеспечивает высокую производительность, кросс – платформенность и обширные возможности для

создания масштабируемого и надежного почтового клиента.

Использование IntelliJ IDEA обеспечивает продвинутые инструменты разработки, существенно улучшая эффективность написания, отладки и тестирования кода. Интегрированная среда разработки предоставляет высокую производительность и комфортное взаимодействие, что является ключевым фактором при работе над развитием почтового клиента.

В связи с необходимостью эффективного управления зависимостями проекта, а также обеспечения порядка и удобства сопровождения, выбор пал на систему управления проектами Maven. Maven автоматизирует процессы сборки, тестирования и управления зависимостями, обеспечивая тем самым легкость в управлении библиотеками.

Кроме того, для реализации пользовательского интерфейса почтового клиента применяются такие технологии, как HTML и CSS. HTML используется для структурирования содержимого веб-страниц, а CSS – для стилизации и форматирования этих страниц, обеспечивая приятный и интуитивно понятный интерфейс для пользователей [16].

3 РАЗРАБОТКА АРХИТЕКТУРЫ ВЕБ – ПРИЛОЖЕНИЯ

3.1 Описание архитектуры

В созданном приложении осуществляется применение архитектурного подхода Model-View-Controller (MVC), который предусматривает явное разделение на три основных компонента: "модели", "виды" и "контроллеры". Этот подход обеспечивает эффективное управление структурой приложения, обеспечивая легкость в сопровождении и масштабировании.

Модели в контексте почтового клиента определяют структуру данных, представляющих сущности в базе данных. Здесь модели включают информацию о пользователях, письмах, вложениях и других ключевых элементах почтового клиента.

Виды представляют данные, которые сервер возвращает клиенту в ответ на запросы. В рамках почтового клиента, виды включают в себя отображение списка писем, содержание конкретного письма, информацию о контактах и другие элементы интерфейса.

Контроллеры в приложении подразделяются на три основных компонента: "rest-контроллеры", "сервисы" и "репозитории". Rest-контроллеры обрабатывают запросы от клиента, определяя параметры для взаимодействия с сервером. Сервисы осуществляют обработку данных, полученных от клиента, используя репозитории и другие вспомогательные сервисы. Репозитории взаимодействуют с базой данных, выполняя SQL-запросы для эффективного управления данными. Эта структура контроллеров обеспечивает четкое разделение обязанностей и управление бизнес-логикой, что способствует более эффективному функционированию приложения "Почтовый клиент". Применение архитектуры MVC также обеспечивает гибкость, удобство в разработке и обслуживании, что является важным аспектом в контексте создания почтового клиента.

На рисунке 3.1.1 представлена схема, используемой архитектуры.

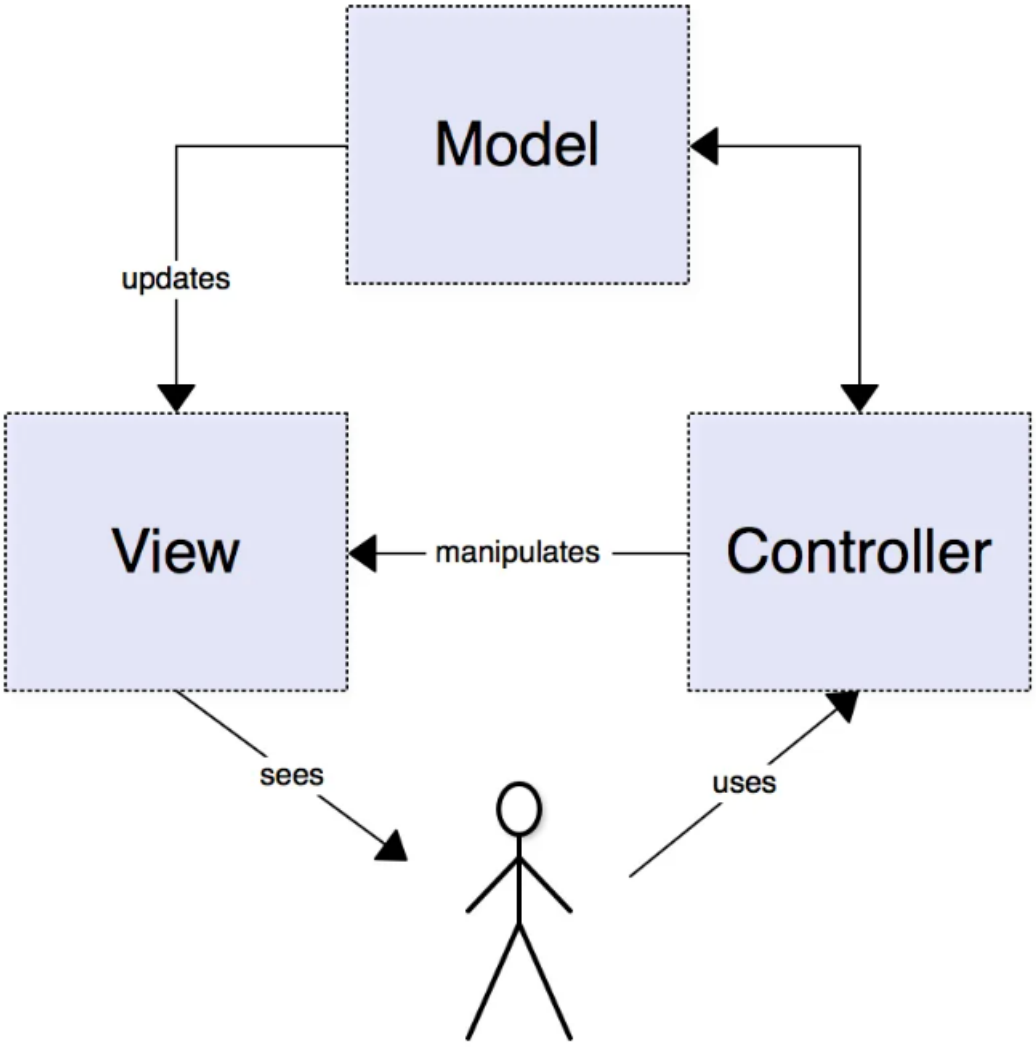

Рисунок 3.1.1 – Архитектура MVC

3.2 Реализация архитектуры

На рисунке 3.2.1 показана схема выбранной архитектуры на примеры получение информации данных клиента почтового ящика.

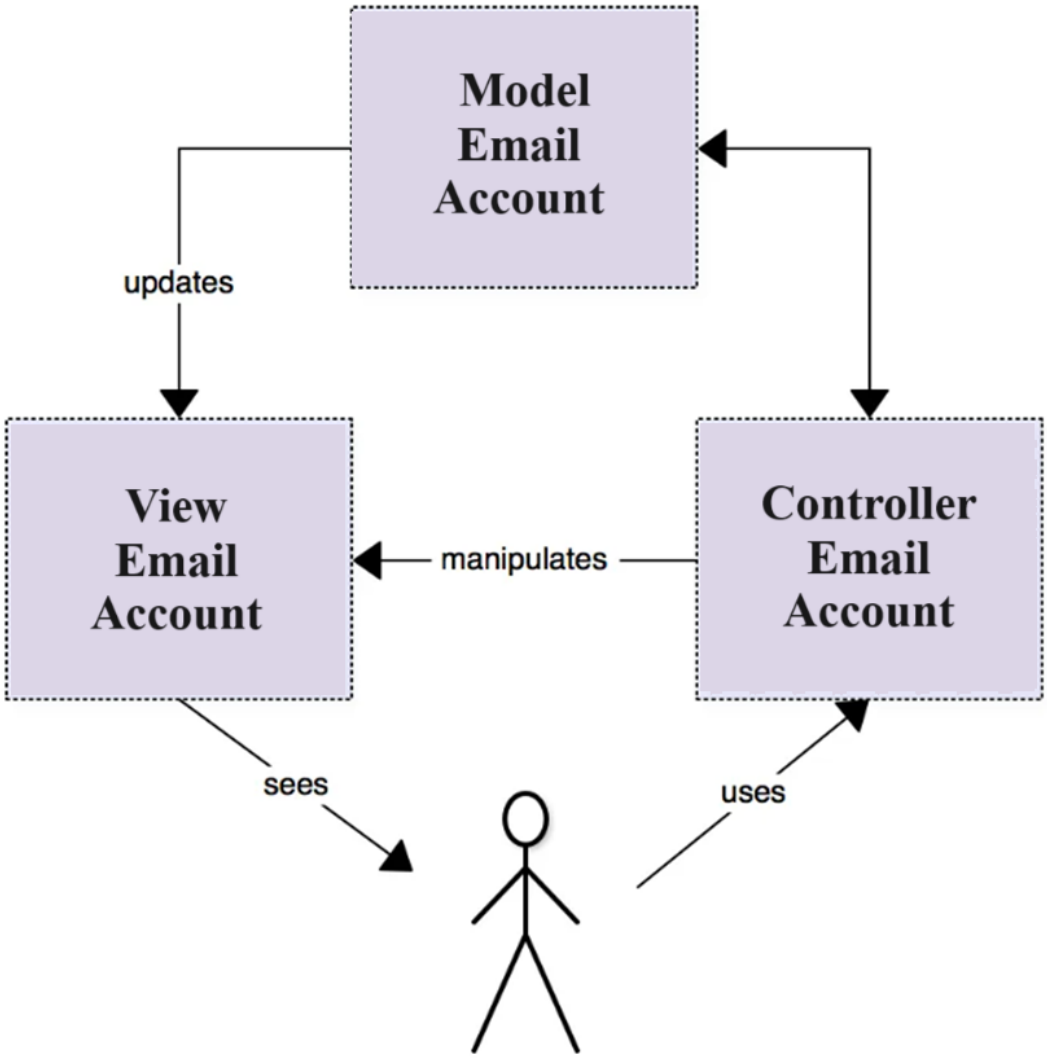

Рисунок 3.2.1 – Использование архитектура MVC

Пользователь делает запрос, обращаясь к rest-контроллеру по определенному эндпоинту. Rest-контроллер, в свою очередь, получает запрос и определяет параметры взаимодействия с сервером, такие как метод HTTP, заголовки и тело запроса – листинг 3.2.1.

Листинг 3.2.1 – Контроллер для работы с данными почтовых аккаунтов

```
@RestController
@RequestMapping("/email")
@RequiredArgsConstructor
public class EmailController {
    private final EmailAccountServices emailAccountServices;
    private final MessageInServices messageInServices;
    private final MessageOutServices messageOutServices;
```

Продолжение листинга 3.2.1

```
@GetMapping("")
    public List<EmailAccount> index(Model model){
        return emailAccountServices.findAll();
    }
    @GetMapping("/{id}")
    public EmailAccount show(@PathVariable("id") int id, Model
model){
        return emailAccountServices.findOne(id);
    }
```

Получив запрос от контроллера, сервис приступает к обработке данных. В данном случае, сервис может включать в себя логику для извлечения информации о почтовых ящиках из базы данных.

Сервис начинает взаимодействовать с репозиториями для получения необходимых данных и проведения дополнительной бизнес-логики, связанной с аккаунтами почтовых ящиков – листинг 3.2.2.

Листинг 3.2.2 – Сервис для работы с данными почтовых аккаунтов

```
@Service
@RequiredArgsConstructor
public class EmailAccountServices implements UserDetailsService{
    private final EmailAccountRepositories
emailAccountRepositories;
    public List<EmailAccount> findAll(){
        return emailAccountRepositories.findAll();
    }
    public EmailAccount findOne(int id){
        return
emailAccountRepositories.findById(id).orElse(null);
    }
```

Репозитории предоставляют интерфейс для взаимодействия с базой данных. В контексте запроса данных аккаунтов почтовых ящиков, репозиторий может выполнять SQL-запросы для извлечения необходимых записей из таблицы, представляющей аккаунты – листинг 3.2.3.

Листинг 3.2.3 – Репозиторий для работы с данными почтовых аккаунтов

```
@Repository
public interface EmailAccountRepositories extends
JpaRepository<EmailAccount, Integer> {
    Optional<EmailAccount> findByEmail(String email);
}
```

В контексте данного запроса, модель представляет собой объекты, экземпляры сущности, которые содержат информацию о почтовых ящиках. Когда сервис и репозиторий успешно обработали запрос, модель возвращается в виде ответа на запрос пользователя – листинг 3.2.4.

Листинг 3.2.4 – Модель, представляет данные об почтовых аккаунтах

```
@Entity
@Table(name="EmailAccount")
@Data
@Builder
@AllArgsConstructor
@RequiredArgsConstructor
public class EmailAccount {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String surname;
    @Column(unique = true)
    private String email;
    private String password;
    @Enumerated(EnumType.STRING)
    private Role role;
```

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Определение сущностей

На основе выявленных функциональных требований и анализа предметной области, предстоит разработать сущности базы данных, которые эффективно отражают структуру данных приложения "Почтовый клиент". Функциональные требования определяют ключевые операции и взаимодействия в приложении, а их анализ позволяет выделить основные сущности и их атрибуты, которые представлены на рисунке 4.1.1. Проектирование базы данных становится важным этапом, поскольку оно обеспечивает основу для хранения и управления данными, необходимыми для полноценного функционирования почтового клиента.

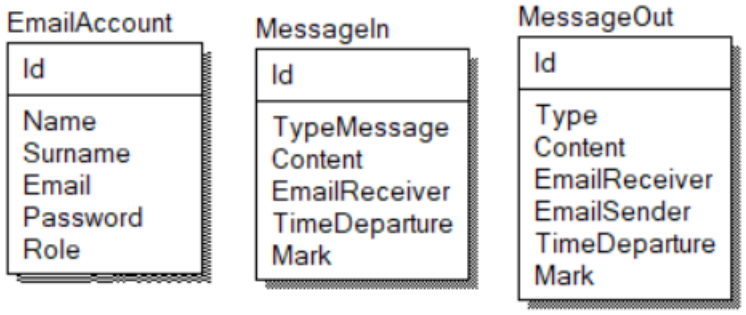

Рисунок 4.1.1 – Структура выявленных сущностей

4.2 Описание сущностей

4.2.1 EmailAccount

Данная сущность представляет собой хранилище информации о пользователях почтового клиента. Каждый экземпляр этой сущности соответствует отдельному аккаунту пользователя, имеющему доступ к функционалу почтового приложения.

Сущность "Email_Account" обеспечивает хранение и управление данными пользовательских аккаунтов в почтовом клиенте. Каждый аккаунт характеризуется уникальным идентификатором, адресом электронной почты, именем, паролем, ролью (пользователь или администратор) и фамилией пользователя. Эта сущность играет ключевую роль в обеспечении аутентификации, авторизации и персонализации пользовательского опыта в приложении.

4.2.2 MessageIn

Сущность "Message_In" отражает входящие сообщения в почтовом клиенте, предоставляя информацию о каждом сообщении, такую как его содержание, адресат, статус метки (прочитано/непрочитано), время отправки, тип сообщения (входящее/исходящее) и связь с конкретным пользовательским аккаунтом. Эта сущность играет важную роль в управлении входящими сообщениями, обеспечивая их эффективное хранение, обработку и связь с пользователями в рамках почтового приложения.

4.2.3 MessageOut

Сущность "Message_Out" аналогична сущности "Message_In" и предназначена для хранения информации об исходящих сообщениях в почтовом клиенте. Каждое сообщение содержит уникальный идентификатор, содержание, адресата, отправителя, метку (прочитано/непрочитано), временную метку отправки, тип (входящее/исходящее) и связанный идентификатор пользователя.

Сущность "Message_Out" подчеркивает структурное сходство с "Message_In", но добавляет информацию об отправителе сообщения, что делает ее полезной для хранения и управления исходящими сообщениями в контексте почтового приложения.

4.3 Проектирование сущностей

На основе предварительного выбора PostgreSQL в качестве основной системы управления базами данных для приложения "Почтовый клиент", проектирование всех сущностей будет вестись именно в этой СУБД, реализация которой представлена на рисунке 4.3.1. Такой подход обеспечивает согласованность и целостность базы данных, учитывая специфику и требования почтового приложения.

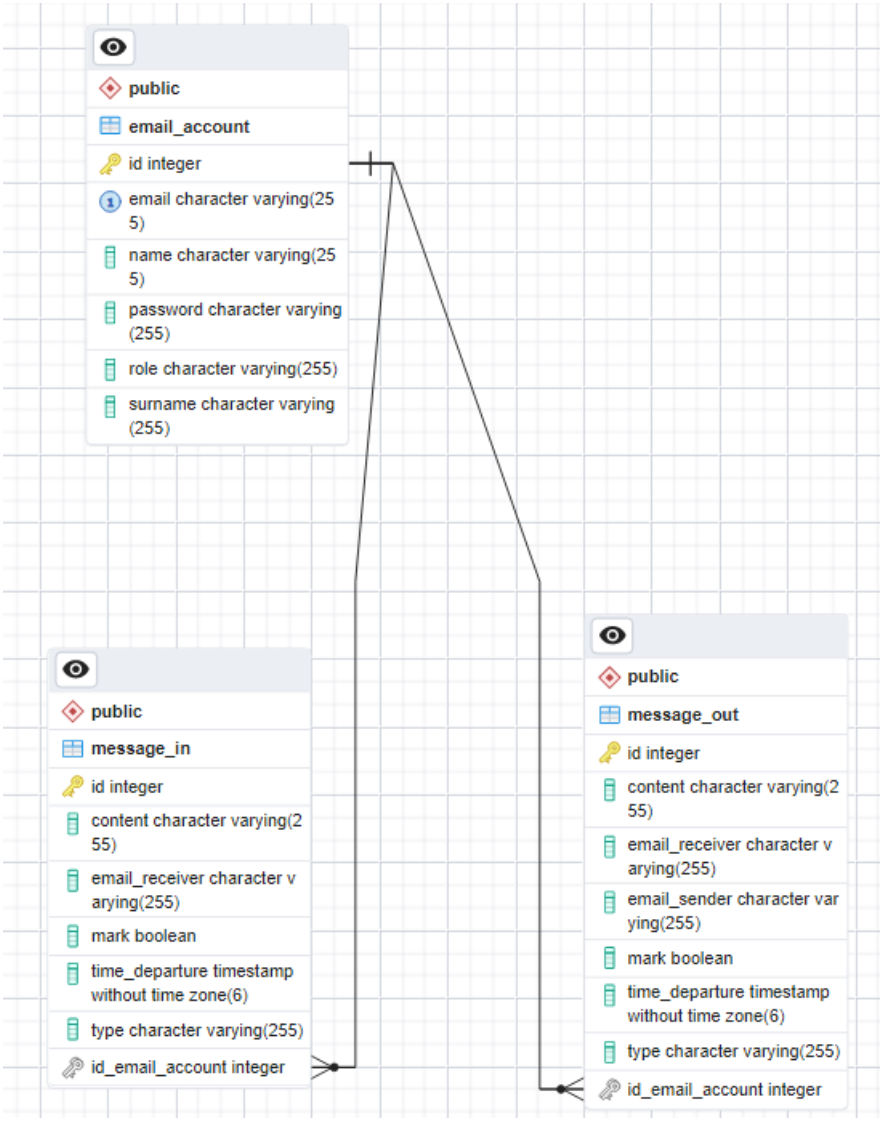

Рисунок 4.3.1 – ER-диаграмма спроектированная выбранной СУБД

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ

5.1 Структура проекта

На рисунке 5.1.1 представлена структура проекта. В основной директории проекта содержатся Java-файлы, необходимые для корректного функционирования приложения. Также присутствуют технические файлы, включая Maven-файл для сборки проекта и конфигурацию для работы в IntelliJ IDEA. Эти компоненты обеспечивают организованность и управление проектом в процессе его разработки и сопровождения.

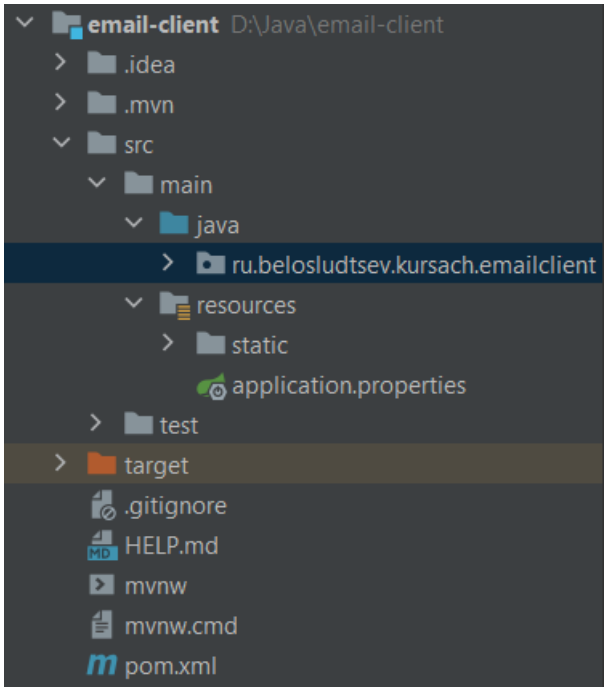

Рисунок 5.1.1 – Структура серверной части проекта

Внутри основной директории "main" проекта можно выделить различные поддиректории, каждая из которых предназначена для конкретных функциональных компонентов почтового приложения – рисунок 5.1.2. В поддиректории "controllers" находятся Java-классы, ответственные за обработку HTTP-запросов и функциональность контроллеров. Директория "services" содержит классы, предоставляющие бизнес-логику и сервисы, необходимые для работы приложения. Репозитории, предназначенные для взаимодействия с базой данных, располагаются в поддиректории "repositories". Модели данных, определяющие структуру объектов, хранятся в директории "models". Кроме того, в поддиректории "config" находятся классы,

занимающиеся конфигурацией и настройками приложения. Эта структура обеспечивает четкое разделение функциональности и удобное управление кодовой базой.

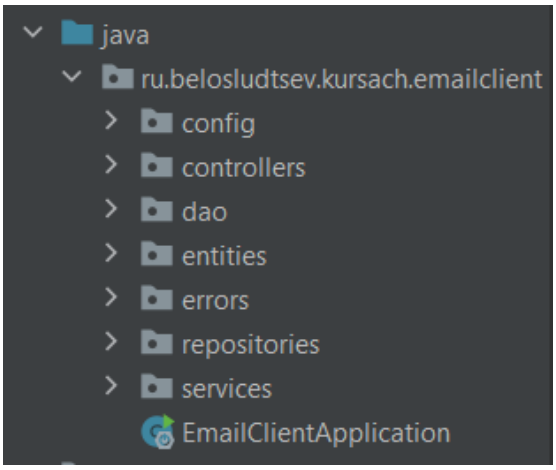

Рисунок 5.1.2 – Структура директории main

5.2 Конфигурация проекта

Файл "application.properties" предназначен для централизованного управления параметрами и настройками приложения "Почтовый клиент". В нем определены параметры подключения к базе данных PostgreSQL, настройки отображения SQL-запросов в консоли, стратегия обновления схемы базы данных, а также секретный ключ для подписи JWT-токенов. Этот файл, представленный на листинге 5.2.1, обеспечивает эффективное и единообразное управление конфигурацией приложения, облегчая его развертывание и сопровождение.

Листинг 5.2.1 – Файл «application.properties»

```
spring.datasource.url=jdbc:postgresql://localhost:5432/EmailCli
ent
spring.datasource.username=postgres
spring.datasource.password=belka2003

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

token.siging.key=404E635266556A586E3272357538782F413F4428472B4B
6250645367566B5970
```

5.3 Авторизация и аутентификация

При анализе предметной области разработки почтового клиента выявлены ключевые аспекты, неотъемлемыми из которых являются механизмы авторизации и аутентификации. Эти аспекты играют критическую роль в обеспечении безопасности и контроля доступа к конфиденциальной информации.

Авторизация – процесс проверки прав доступа пользователя к ресурсам системы, в то время как аутентификация подразумевает подтверждение личности пользователя. Эти процессы становятся ключевыми факторами в почтовом клиенте, где важно обеспечить защиту личных данных и предотвратить несанкционированный доступ.

Реализация механизма авторизации и аутентификации в данном приложении будет произведена с использованием JWT-токенов. JWT (JSON Web Token) представляет собой стандарт для обмена данными в зашифрованном формате между сторонами. Его применение позволяет безопасно и эффективно подтверждать подлинность пользователей и предоставлять им доступ к функциональности почтового клиента.

Этот механизм реализуется с использованием Spring Security, мощного инструмента для обеспечения безопасности в приложениях на платформе Spring. Spring Security предоставляет гибкие средства для конфигурации аутентификации и авторизации, что делает его идеальным выбором для обеспечения безопасности в почтовом клиенте [17].

5.3.1 Класс «SecurityConfiguration»

Класс SecurityConfiguration представляет собой конфигурацию системы безопасности для веб-приложения "Почтовый клиент" – листинг 5.3.1.1. Обозначен аннотациями @Configuration и @EnableWebSecurity, что свидетельствует о его роли в настройке безопасности и обеспечении защиты веб-ресурсов. Класс реализует интерфейс SecurityConfigurerAdapter, который позволяет настраивать параметры безопасности приложения.

В методе securityFilterChain определены основные настройки

безопасности, такие как разрешение доступа к определенным ресурсам (например, эндпоинтам для аутентификации), управление сессиями (в данном случае, STATELESS), а также добавление фильтра для обработки JWT-токенов (jwtAuthenticationFilter), предназначенного для проверки подлинности пользователей.

Метод authenticationProvider конфигурирует объект DaoAuthenticationProvider, который отвечает за проведение аутентификации пользователей. В данном случае, используется сервис emailAccountServices для загрузки информации о пользователях, а также passwordEncoder() для безопасного хранения и проверки паролей.

PasswordEncoder инстанциируется в методе passwordEncoder() и используется для хеширования паролей пользователей с помощью алгоритма BCrypt.

Также в классе определен бин authenticationManager, который предоставляет объект AuthenticationManager для обработки запросов аутентификации.

Листинг 5.3.1.1 – Класс «SecurityConfiguration»

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {
    private final JwtAuthenticationFilter
jwtAuthenticationFilter;
    private final EmailAccountServices emailAccountServices;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception{
        http
                .cors(Customizer.withDefaults())
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(request ->
request.requestMatchers("/auth/**")

.permitAll().anyRequest().authenticated())
                .sessionManagement(manager ->
manager.sessionCreationPolicy(STATELESS))

.authenticationProvider(authenticationProvider()).addFilterBefor
e(
                        jwtAuthenticationFilter,
```

Продолжение листинга 5.3.1.1

```
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
@Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
DaoAuthenticationProvider();

authProvider.setUserDetailsService(emailAccountServices);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config)
            throws Exception {
        return config.getAuthenticationManager();
    }
}
```

5.3.2 Класс «JwtAuthenticationFilter»

JwtAuthenticationFilter представляет собой компонент конфигурации Spring Security, специализированный в обработке JWT-токенов и проведении аутентификации пользователей. Этот класс реализует интерфейс OncePerRequestFilter, обеспечивающий выполнение фильтрации только один раз для каждого HTTP-запроса – листинг 5.3.2.1.

Внутри метода doFilterInternal, который реализует обработку запроса, выполняются следующие шаги:

Извлекается заголовок "Authorization" из HTTP-запроса.

Если заголовок пуст или не начинается с "Bearer ", фильтр пропускает запрос дальше, не вмешиваясь в процесс.

Если токен успешно извлечен, из него извлекается электронная почта пользователя.

Если электронная почта не является пустой и аутентификация еще не была выполнена (текущий контекст без аутентификации), загружаются детали пользователя из сервиса пользователей, и проверяется валидность токена.

В случае валидного токена создается аутентификационный объект UsernamePasswordAuthenticationToken, содержащий информацию о пользователе и его ролях.

Аутентификационный объект устанавливается в текущий контекст безопасности SecurityContextHolder. Запрос передается дальше в цепочку фильтров для обработки.

Этот фильтр обеспечивает проверку JWT-токена и автоматическую аутентификацию пользователя на основе этого токена при каждом HTTP-запросе. Такой механизм гарантирует безопасность и удобство взаимодействия с защищенными ресурсами приложения.

Листинг 5.3.2.1 – Класс «JwtAuthenticationFilter»

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends
OncePerRequestFilter {
    private final JwtServices jwtService;
    private final EmailAccountServices emailAccountService;
    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest
request,
                                    @NonNull HttpServletResponse
response,
                                    @NonNull FilterChain
filterChain)
            throws ServletException, IOException {
        final String authHeader =
request.getHeader("Authorization");
        final String jwt;
        final String userEmail;
        if (StringUtils.isEmpty(authHeader) ||
!StringUtils.startsWith(authHeader, "Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring(7);
userEmail = jwtService.extractUserName(jwt);
        if (StringUtils.isNotEmpty(userEmail)
        &&
SecurityContextHolder.getContext().getAuthentication() == null)
{
        UserDetails userDetails =
emailAccountService.loadUserByUsername(userEmail);
```

Продолжение листинга 5.3.2.1

```
            if (jwtService.isTokenValid(jwt, userDetails)) {
                SecurityContext context =
SecurityContextHolder.createEmptyContext();
                UsernamePasswordAuthenticationToken authToken =
new UsernamePasswordAuthenticationToken(
                        userDetails, null,
userDetails.getAuthorities());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                context.setAuthentication(authToken);
                SecurityContextHolder.setContext(context);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

5.4 Реализация основного функционала

Одним из ключевых требований при разработке почтового клиента была не только надежная система авторизации и аутентификации с использованием JWT-токенов, но и обеспечение высокого уровня удобства для пользователя. В рамках предметной области выделены важные аспекты, такие как удобный просмотр, эффективная фильтрация и организация почтовых сообщений. Эти аспекты стали основой для разработки функционала, который обеспечивает интуитивно понятный и эффективный опыт взаимодействия с приложением.

Теперь переходим к реализации данных функциональных требований в соответствующих классах. Каждый класс направлен на обеспечение определенного аспекта управления и просмотра почтовыми сообщениями, при этом придерживаясь принципов удобства использования и высокой производительности.

5.4.1 Отправка и получение сообщений

Сразу после успешной авторизации открывается расширенный функционал, включая возможность написания и отправки писем. Пользователь, взаимодействуя с веб-интерфейсом – листинг 5.4.1.1, инициирует процесс написания нового письма, отправляя запрос на соответствующий контроллер и эндпоинт.

Листинг 5.4.1.1 – Обработка запроса клиента в контроллере

```
@PostMapping("/{id}/message")
public ResponseEntity<String> sendMessage(@PathVariable("id")
int id,
                                          @RequestBody MessageIn
messageIn) throws EmailReceiverIsNotFound {
    try{
        messageInServices.save(id, messageIn);
        messageOutServices.save(id, messageIn);
        return ResponseEntity.ok("Сообщение успешно
отправлено");

    } catch (EmailReceiverIsNotFound e){
        return ResponseEntity.badRequest().body("Ошибка: " +
e.getMessage());
    }
}
```

На сервере контроллер обрабатывает запрос, взаимодействуя с технологиями Spring Framework, такими как Spring MVC для обработки HTTP-запросов и Spring Data JPA для взаимодействия с базой данных – листинг 5.4.1.2.

Листинг 5.4.1.2 – Обработка бизнес-логики в сервисе

```
@Transactional
public void save(int id, MessageIn message) throws
EmailReceiverIsNotFound {
    if(!isEmailReceiverExists(message.getEmailReceiver())){
       throw  new EmailReceiverIsNotFound("Email получателя не
существует");
    }
    saveIn(id,message);
    messageInRepositories.save(message);
}

private boolean isEmailReceiverExists(String emailReceiver) {
    EmailAccount emailAccount =
emailAccountServices.findByEmail(emailReceiver);
    if (emailAccount != null) return true;
    else return false;
}
private void saveIn(int id, MessageIn message) {
    EmailAccount emailAccount =
emailAccountServices.findOne(id);
    message.setType(TypeMessage.IN_BOX);
    message.setTimeDeparture(LocalDateTime.now());
    message.setSender(emailAccount);
    emailAccount.addMessageIn(message);
}
```

В ходе обработки запроса сервер инициализирует две сущности: MessageIn и MessageOut, представляющие входящие и исходящие письма соответственно. Данные для этих сущностей передаются с веб-клиента в виде параметров запроса или тела запроса.

Следующим шагом созданные сущности привязываются к соответствующим сущностям emailAccount отправителя и получателя с использованием Spring Data JPA – листинг 5.4.1.3.

Листинг 5.4.3.1 – Установка связей между сущностями

```
public void addMessageIn(MessageIn message){
    if(this.messagesIn == null){
        messagesIn = new ArrayList<>();
    }
    messagesIn.add(message);
}
public void addMessageOut(MessageOut message){
    if(this.messagesOut == null){
        messagesOut = new ArrayList<>();
    }
    messagesOut.add(message);
}
```

Затем происходит обновление списков сообщений в аккаунтах отправителя и получателя. Новые сообщения добавляются к списку отправленных сообщений в аккаунте отправителя и к списку полученных сообщений в аккаунте получателя.

В результате успешного выполнения операций пользователь получает обратную связь об успешной отправке письма, реализованную с использованием HTTP-статусов ответа и соответствующих сообщений об успешном выполнении операции.

Таким образом, весь процесс отправки писем охвачен с использованием современных технологий веб-разработки, включая Spring Framework, Spring Data JPA и механизмы безопасности Spring Security.

5.4.2 Сортировка и фильтрация сообщений

После успешной аутентификации пользователя в системе, каждой из сущностей emailAccount открывается возможность просмотра своих сообщений. Эта функциональность реализована через взаимодействие с веб-

интерфейсом и использование HTTP-запросов к определенному эндпоинту.

Пользователь, обращаясь к соответствующему эндпоинту, может указать дополнительные параметры в запросе, используя аннотацию «request param». Эти параметры определяют характер последующей бизнес-логики в сервисах – листинг 5.4.2.1.

Листинг 5.4.2.1 – Обработка запрос клиента в контроллере

```
@GetMapping("/{id}/getMessagesIn")
public List<MessageIn> getMessagesIn(@PathVariable("id") int id,
                                     @RequestParam(value="mark",
required = false) Boolean mark,

@RequestParam(value="sortedByName", required = false) Boolean
sort){
    return emailAccountServices.findMessagesIn(id, mark, sort);
}
```

Возможности сортировки и фильтрации сообщений предоставляют пользователю гибкий механизм настройки отображения своей почты. На основе переданных параметров, например, сортировка по дате, отправителю или получателю, сервисы принимают решения о том, как предоставить данные для отображения.

Пользователь может также указать дополнительные фильтры, такие как отображение только важных сообщений. Эти параметры также передаются в запросе, и на основе них сервисы проводят поиск и формируют ответ с соответствующими данными – листинг 5.4.2.2.

Листинг 5.4.2.2 – Обработка бизнес-логики в сервисе

```
public List<MessageIn> findMessagesIn(int id, Boolean mark,
Boolean sort){
    if(mark == null && (sort == null || !sort) ){
        return findMessageIn(id);
    }
    else if(mark != null && (sort == null || !sort) ){
        return findMessageIn(id, mark);
    }
else if(mark == null && sort){
        return emailAccountRepositories.findById(id)
                .map(emailAccount ->
emailAccount.getMessagesIn()
                 .stream()
.sorted(Comparator.comparing(MessageIn::getEmailReceiver))
```

Продолжение листинга 5.4.2.2

```
                                .collect(Collectors.toList()))
                .orElse(null);
    }
    else{
        return emailAccountRepositories.findById(id)
                .map(emailAccount ->
emailAccount.getMessagesIn()
                .stream()
                .filter(messageIn -> messageIn.isMark() == mark)

.sorted(Comparator.comparing(MessageIn::getEmailReceiver))
                .collect(Collectors.toList()))
                .orElse(null);
    }

}
```

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ

6.1 Структура проекта

В директории проекта находятся разнообразные технические файлы и каталоги, включая файлы для настройки NodeJS, конфигурации ESLint с интегрированным Prettier, а также настройки сборщика Vite. В дополнение к этому, там присутствуют модули для всех использованных библиотек, рабочий каталог проекта (src) и папка, содержащая собранный результат приложения (dist). Весь этот набор можно визуально оценить на рисунке 6.1.1.

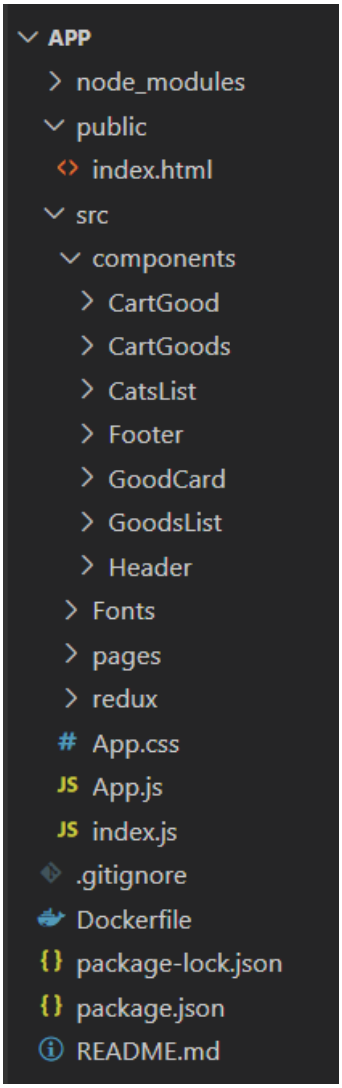

Рисунок 6.1.1 – Структура клиентской части проекта

6.2 Страницы разработанного приложения

На рисунках 6.2.1 – 6.2.4 представлены разработанные страницы клиентской части проекта. Главная страница содержит основную информацию, включая общий обзор аккаунта пользователя и его действий.

Страница авторизации предоставляет возможность ввода учетных данных для входа [18]. Страница входящих сообщений демонстрирует полученные сообщения с удобными опциями фильтрации и сортировки. Страница отправленных сообщений позволяет отслеживать отправленные сообщения. Страница содержимого сообщения предоставляет детализированное представление конкретного сообщения с текстом и прикрепленными файлами.

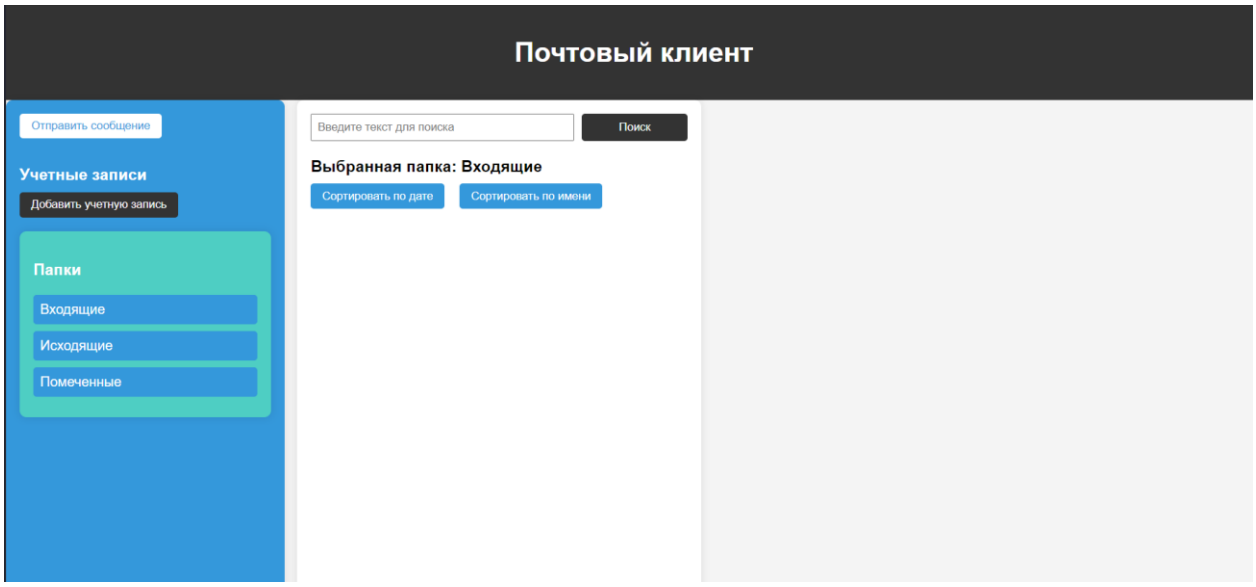

Рисунок 6.2.1 – Главная страница

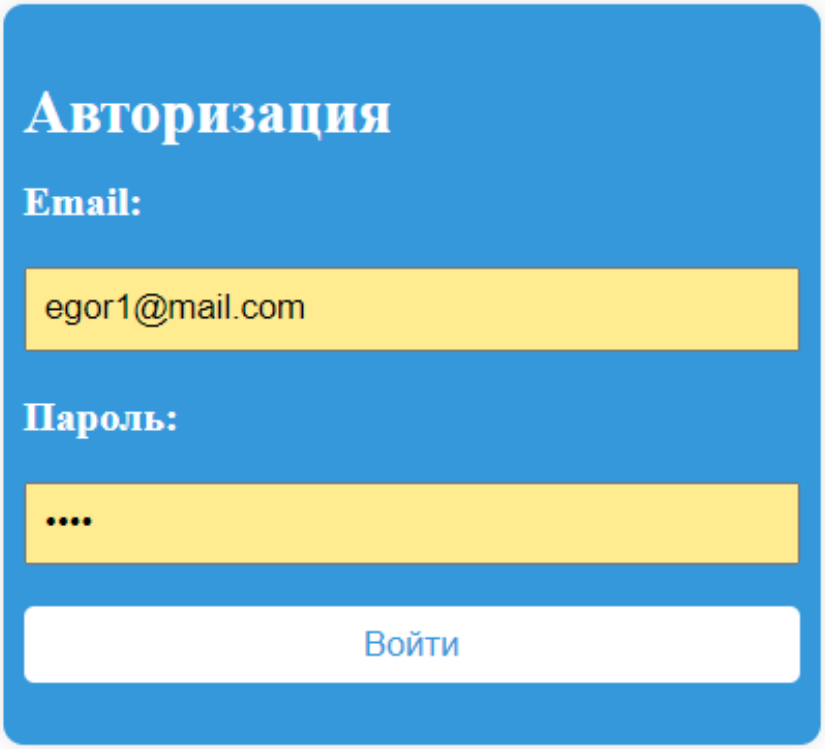

Рисунок 6.2.2 – Страница авторизации

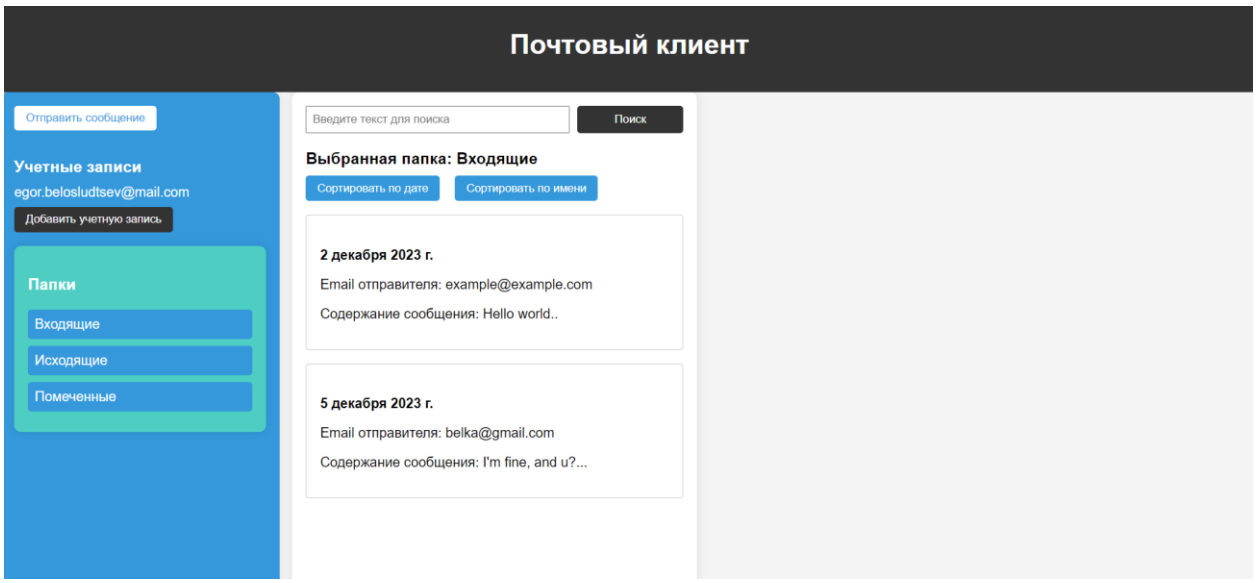

Рисунок 6.2.3 – Страница пользователя

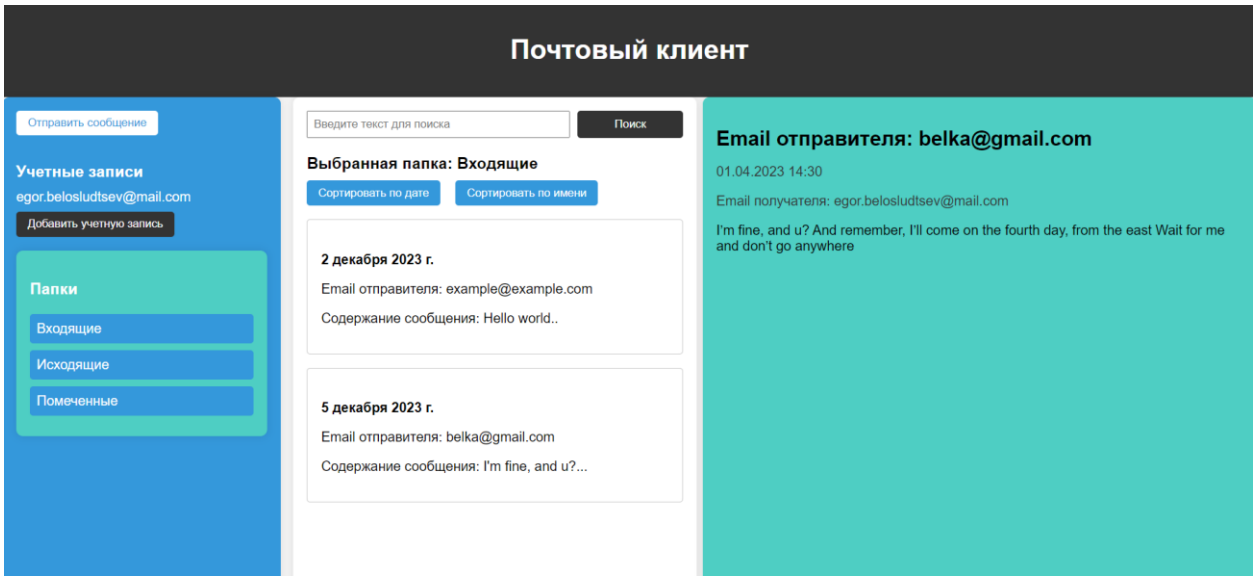

Рисунок 6.2.4 – Страница просмотра сообщения у пользователя

7 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

7.1 Тестирование приложения с помощью Postman

Протестируем работы регистрации новой учетной записи почтового ящика. Для этого выполним POST запрос, указав в теле запроса персональные данные – рисунок 7.1.1.

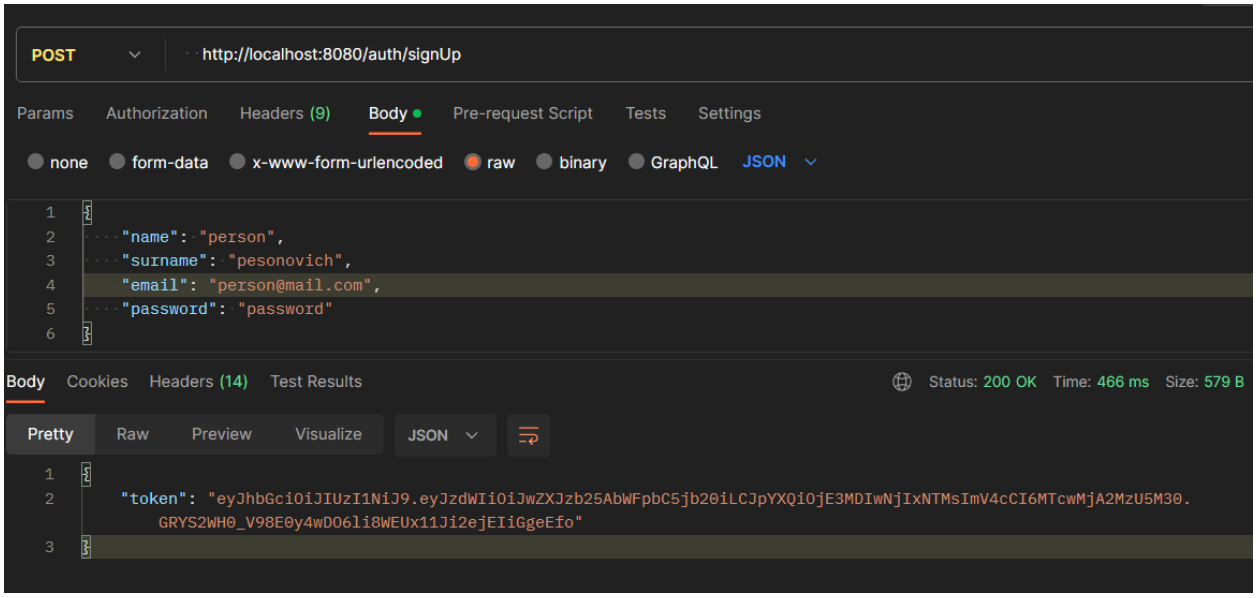

Рисунок 7.1.1 – POST запрос на создание новой учетной записи

В результате был получен сообщение со статусом «200», что подтверждаем корректность выполнение данного запроса, а также в теле ответа был записан токен, который нужен для дальнейшей работы с этим аккаунтом.

Дальше протестируем выполнение GET запроса, который позволит получить информацию об данной учетной записи – рисунок 7.1.2. Для только, чтобы получить к этим данным, нужно не забыть вставить токен, полученный после регистрации (авторизации), в тело запроса.

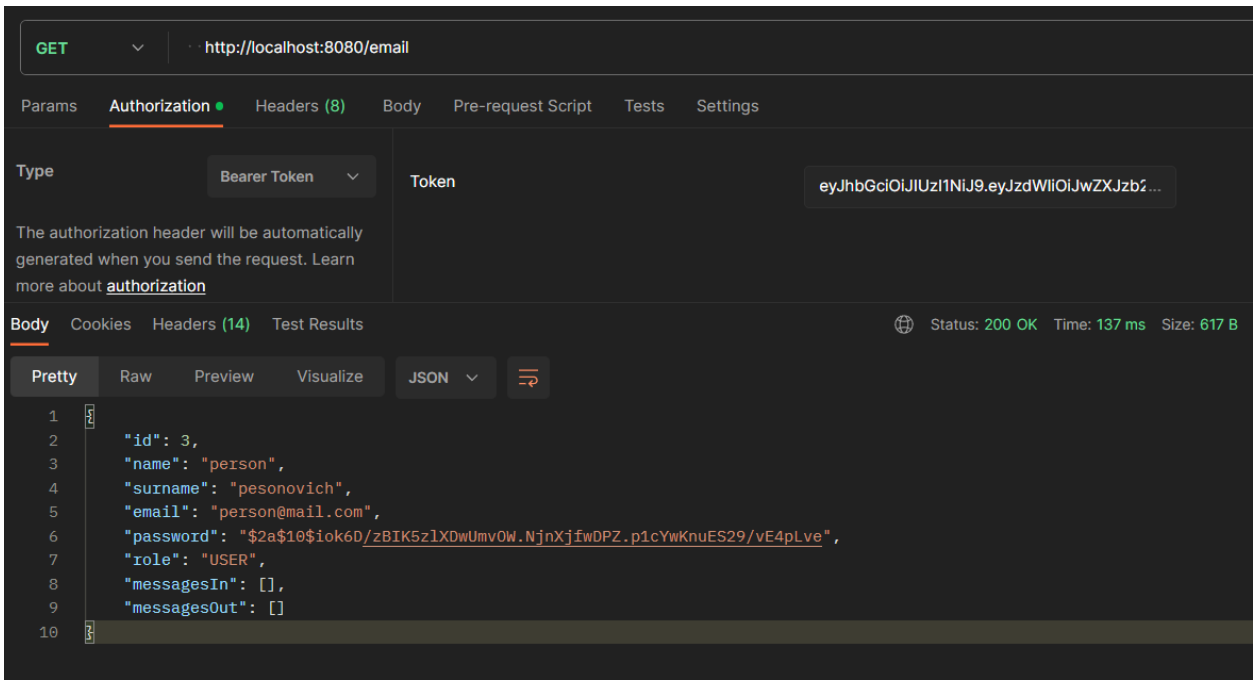

Рисунок 7.1.2 – GET запрос на получение данных об учетной записи

Теперь протестируем корректность выполнения POST запроса для отправления письма – рисунок 7.1.3.

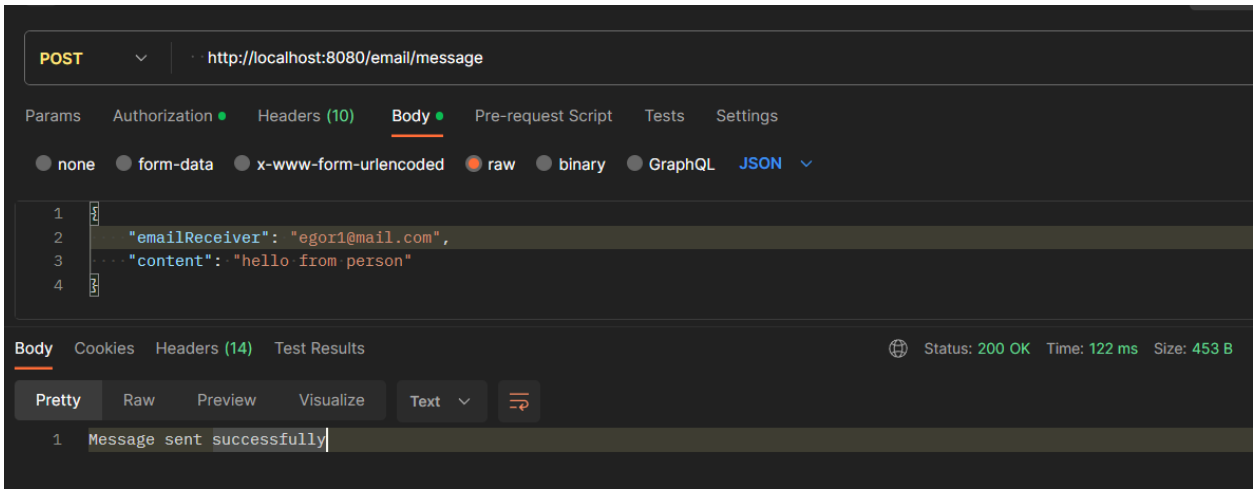

Рисунок 7.1.3 – POST запрос на создание нового сообщения

Чтобы окончательно убедиться, что сообщения было доставлено до нужного адресанта, для этого выполним GET запрос, предварительно авторизовавшись. Результат представлен на рисунке 7.1.4.

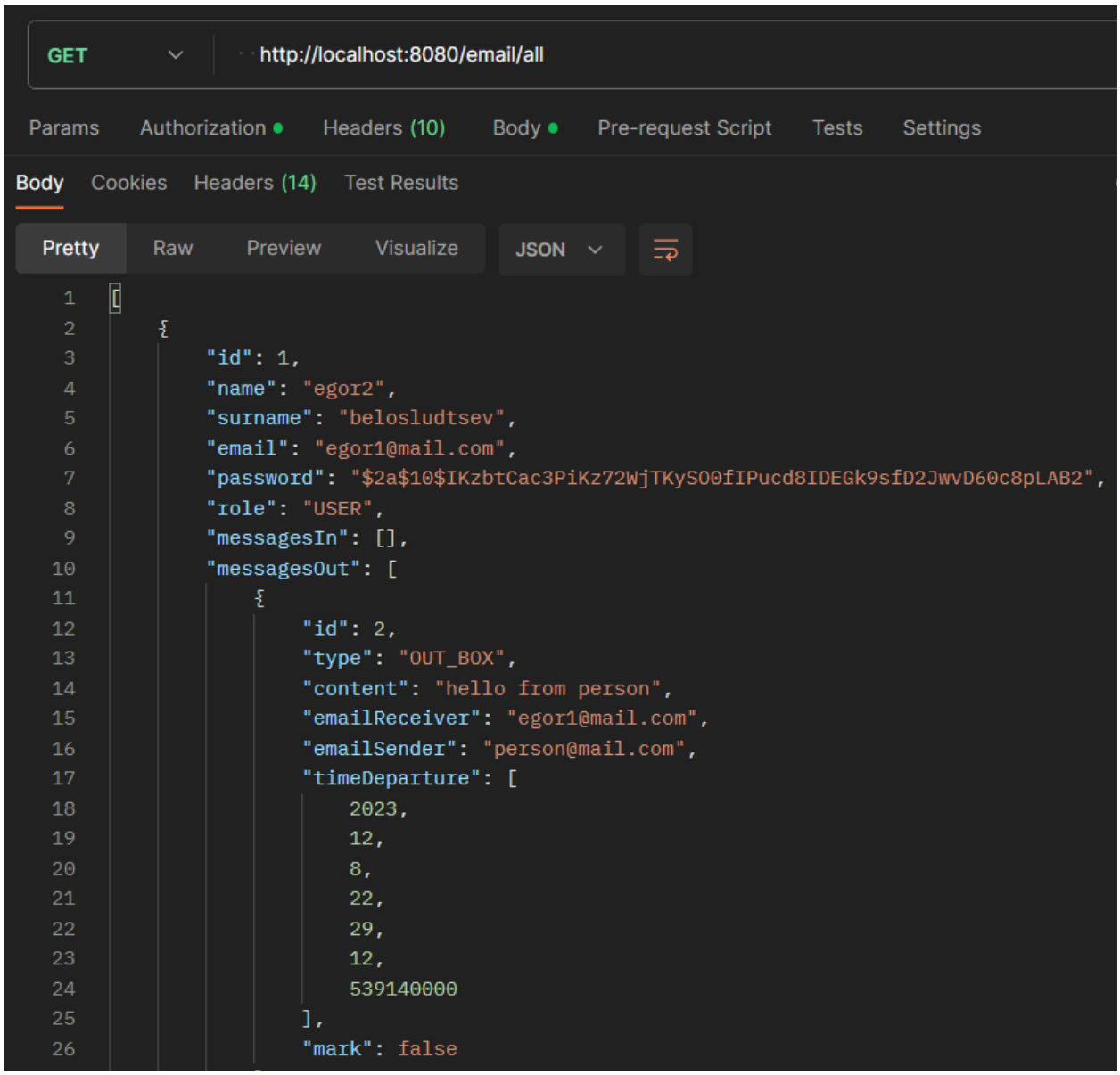

Рисунок 7.1.4 – GET запрос на получение данных об учетной записи

Как видно из результата список полученных сообщений пополнился одной записью, то есть функцию отправки сообщений работает корректно.

Также осталось продемонстрировать GET запрос на получение сообщений, указываю различные данные в заголовке запроса. Результаты получения сообщений по дате продемонстрирован на рисунке 7.1.5, а на рисунке 7.1.6 – результат получения сообщений по имени получателя.

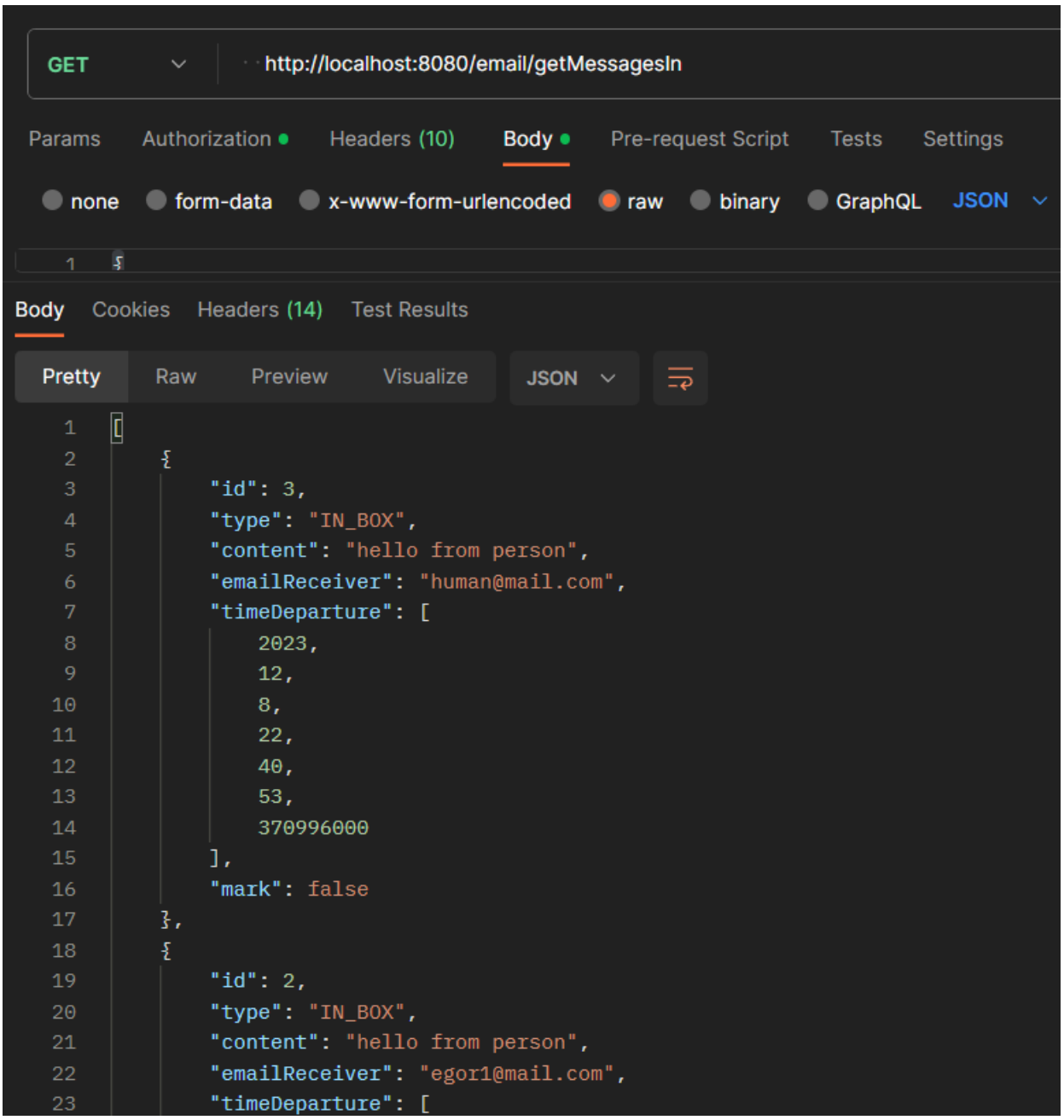

Рисунок 7.1.5 – GET запрос на получение всех оправленных сообщений, сортируя по дате

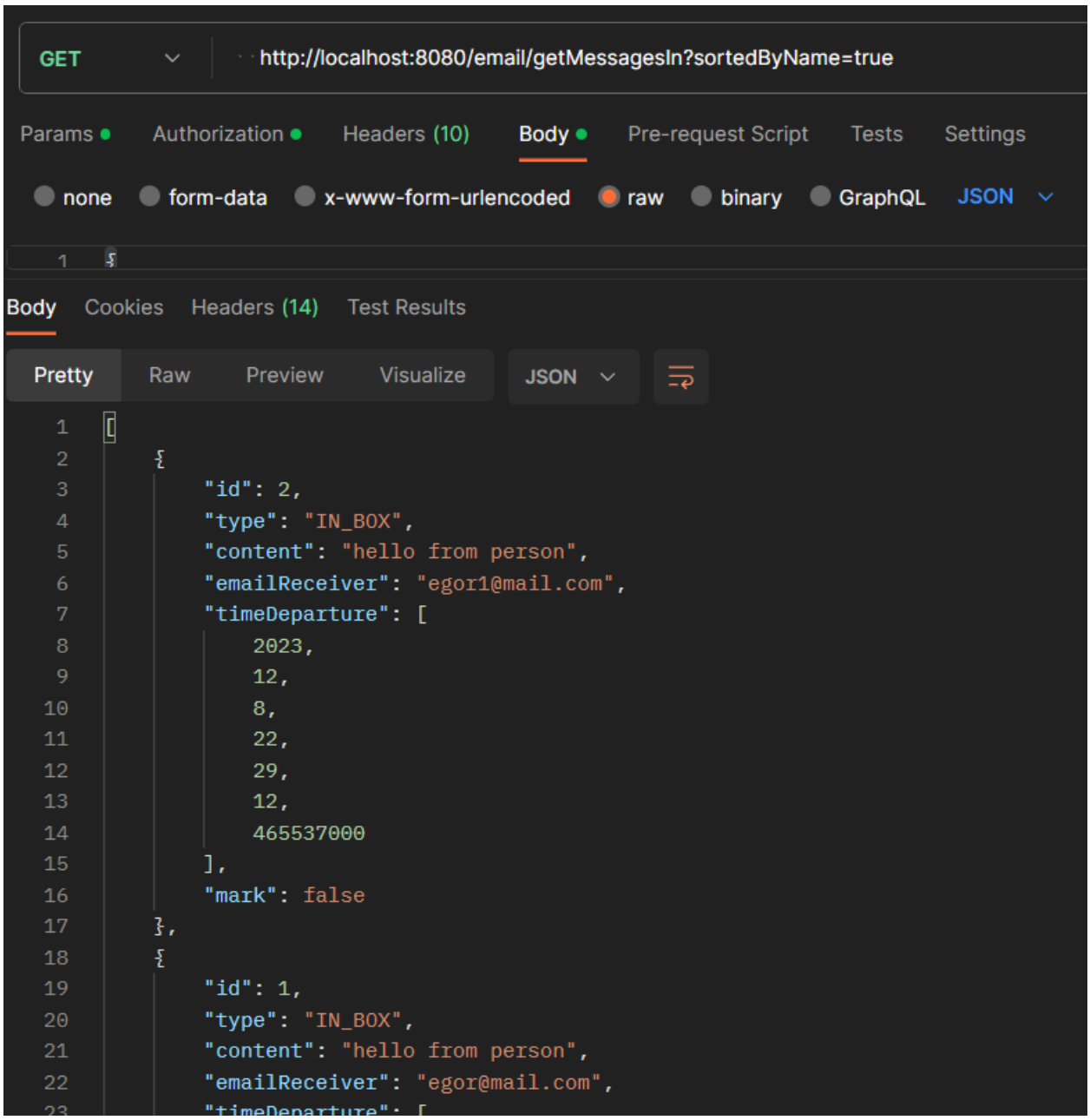

Рисунок 7.1.6 – GET запрос на получение всех оправленных сообщений, сортируя по имени получателя

ЗАКЛЮЧЕНИЕ

Результатом выполнения курсовой работы по теме "Email-клиент с возможностью контроля пересылаемых сообщений" стало разработанное серверное приложение, реализованное с использованием языка программирования Java и фреймворка Spring [19]. В ходе работы использовались ключевые компоненты Spring, такие как Spring Data JPA, Spring Security, Spring Core и Spring MVC. При разработке серверного приложения были активно применены навыки работы в среде разработки IntelliJ IDEA [20].

В рамках курсовой работы углубленно изучены возможности фреймворка Spring, предназначенного для создания Java-приложений. Он предоставил широкий спектр инструментов, существенно упростивших процесс разработки, повысивших производительность и обеспечивших масштабируемость приложения. Также освоена библиотека JavaScript – React, позволяющая создавать динамически управляемые компоненты для интерфейса.

Глубокий анализ предметной области позволил учесть и решить все проблемы, характерные для веб-ресурсов с схожей тематикой. Разработанный веб-ресурс обладает всеми необходимыми элементами, такими как текст и единообразное визуальное оформление, обеспечивая при этом полную функциональность.

Учитывая выполнение поставленных задач, можно утверждать, что все требования, предъявленные курсовой работой, успешно соблюдены. Разработанное приложение представляет собой полноценный и функциональный инструмент для отслеживания и управления пересылаемыми сообщениями в электронной почте.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Руководство по Spring MVC [Электронный ресурс] – URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 22.11.2023).

2. Gmail: почтовый клиент Google. – URL: https://mail.google.com (дата обращения: 20.11.2023).

3. Outlook: почтовый клиент Microsoft. – URL: https://outlook.live.com (дата обращения: 29.11.2023).

4. eMClient: клиент для управления электронной почтой. – URL: https://www.emclient.com (дата обращения: 23.11.2023).

5. Уоллс К. Spring в действии – М.: ДМК Пресс, 2013. – 752 с

6. Документация Spring Boot [Электронный ресурс] – URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата обращения: 20.11.2023).

7. Руководство по Spring Data JPA [Электронный ресурс] – URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (дата обращения: 25.11.2023).

8. Руководство по Spring Security [Электронный ресурс] – URL: https://spring.io/guides/gs/securing-web/ (дата обращения: 24.11.2023).

9. Видео – курс на YouTube «Spring Security Tutorial» [Электронный ресурс] – URL: https://www.youtube.com/watch?v=b9O9NI-RJ3o&t=2825s (дата обращения: 28.11.2023).

10. Руководство по использованию JSON Web Tokens (JWT) [Электронный ресурс] – URL: https://habr.com/ru/articles/533868/ (дата обращения: 01.12.2023).

11. Spring Security JWT Tutorial [Электронный ресурс] – URL: https://www.toptal.com/spring/spring-security-tutorial (дата обращения: 01.05.2023).

12. Руководство по создания RESTful API на Java [Электронный ресурс] – URL: https://spring.io/guides/tutorials/rest/ (дата обращения: 27.11.2023).

13. Руководство по тестированию RESTful API на Medium [Электронный ресурс] – URL: https://medium.com/@oyetoketoby80/how-to-write-unit-test-for-your-rest-api-f8f71376273f (дата обращения: 29.11.2023).

14. Руководство по PostgreSQL [Электронный ресурс] – URL: https://www.postgresql.org/docs/ (дата обращения: 29.11.2023).

15. Мардан А. React быстро. Веб-приложения на React, JSX, Redux и GraphQL – СПб.: Питер, 2019. – 560 с.

16. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 5-е изд.. – СПб.: Питер, 2021 – 768 с.

17. Хоффман Э. Безопасность веб-приложений – СПб.: Питер, 2021. – 354 с.

18. Диков А. Клиентские технологии веб-дизайна. HTML5 и CSS3. Учебное пособие для вузов. 2-е изд.. – Санкт-Петербург: Лань, 2023 – 188 с.

19. Документация Spring Framework [Электронный ресурс]. – URL: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html (дата обращения: 20.11.2023).

20. Исходный код Интернет-ресурса по курсовой работе исполнителя работы: https://github.com/Belchonok31/client-email. – URL: https://github.com/Belchonok31/client-email (дата обращения: [вставьте дату]).