МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств ВТ и автоматизированных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Система приема и отправки почтовых сообщений

Студент 03.06.20 (подпись, дата) Кожаев Вячеслав Дмитриевич (ФИО полностью)

Руководитель (подпись, дата) А.В. Дубравин (фамилия, инициалы)

Нормоконтролёр (подпись, дата) А.В. Кучин (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 20.06.2020 г. № 12)

Заведующий кафедрой (подпись) М.А. Митрохин (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 23.06.2020 № 16)

Секретарь ГЭК (подпись) Е.И. Гурин (фамилия, инициалы)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
Вычислительная техника

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
30 апреля 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Кожаева Вячеслава Дмитриевича гр. 16ВВ2 факультета вычислительной техники.

2.Тема работы: «Система приема и отправки почтовых сообщений».

Тема утверждена приказом ПГУ № 0335 а/о от "30" апреля 2020г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры «ВТ» Дубравин А.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.).
Разработать приложения, приема и отправки почтовых сообщений. Сохранять сообщения и данные пользователей. Разработать механизм приёма и отправки почтовых сообщений. Разработка Rest API и страницы браузера для клиента.

5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметных областей: изучить типы СУБД для хранения данных, виды архитектуры для веб-приложений и рассмотреть основные языки для веб-разработки. Необходимо изучить методологии проектирования веб-приложений, а также изучить функционал фреймворка Spring.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	30.04.2020 – 03.05.2020	
2.	Анализ требований	10	03.05.2020 – 05.05.2020	
3.	Изучение технических средств решения поставленной задачи	15	05.05.2020 – 08.05.2020	
4.	Разработка и тестирование приложения	45	08.05.2020 – 31.05.2020	
5.	Оформление пояснительной записки	10	31.05.2020 – 05.06.2020	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы _____ Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент _____ 30.04.20 Кожаев В.Д.
(подпись, дата)

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

РЕФЕРАТ

Бакалаврская работа содержит 86 листов, 11 рисунков, 10 листингов, 9 источников, 3 приложения.

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, БД, СУБД, REST API, JAVA, SPRING, VUE, VUETIFY.

Цель работы – спроектировать и разработать систему приема и отправки почтовых сообщений.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: прием сообщений, хранение информации о пользователях, хранение сообщений, отправка сообщений, создание API для получения информации на странице браузера.

В результате проведенной работы было разработано приложение, удовлетворяющее поставленной задаче, а именно: разработана структура БД, создан REST API, создана страница браузера. Приложение было протестировано и успешно отлажено.

Содержание

1ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....7

2ВВЕДЕНИЕ.....8

31 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....9

41.1 Анализ типов СУБД.....9

51.2 Выбор реляционной СУБД.....10

61.3 Выбор языка программирования.....14

71.4 Описание архитектуры REST API 16

81.5 Выбор фреймворка для Frontend разработки.....19

91.6 Выбор системы управления версиями Java.....24

10 1.7 Выбор пакетного менеджера.....25

111.8 Описание серверного механизма Thymeleaf.....27

121.9 Описание фреймворка Spring.....29

131.10 Выводы.....30

142 РЕАЛИЗАЦИЯ.....31

152.1 Описание REST API.....32

162.2 Структура БД 35

172.3 Реализация авторизации пользователя. 36

182.4 Реализация клиентской части программы.....37

192.5 Взаимодействие классов.....39

202.6 Программная реализация запроса к БД.....41

213 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....43

224 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ.....49

23ЗАКЛЮЧЕНИЕ.....52

24СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....53

26ПРИЛОЖЕНИЯ.....54

27Приложение А Листинги программы на JavaScript.....54

28Приложение В Листинг FrontEnd.....70

29Приложение D Презентация.....81

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API (Application Programming Interface) – программный интерфейс приложения;

HTML (HyperText Markup Language) – язык гипертекстовой разметки;

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста;

ID (Identifier) – идентификатор;

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

NoSQL (not only SQL) – не только SQL;

OLAP (Online Analytical Processing) – интерактивная аналитическая обработка;

REST (Representational State Transfer) – передача состояния представления;

SQL (Structured Query Language) – язык структурированных запросов;

UML (Unified Modeling Language) – унифицированный язык моделирования;

URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса;

XML (eXtensible Markup Language) – расширяемый язык разметки;

БД – База данных;

ОРСУБД – объектно-реляционная система управления базами данных;

РСУБД – система управления реляционными базами данных;

СУБД – система управления базами данных.

ВВЕДЕНИЕ

Электронная почта (E-mail) является одной из самых первых технологий для передачи и получения сообщений по компьютерной сети. Зародившись в далеком 1965 году, она стала революцией в сфере коммуникаций и дала толчок к развитию сети интернет.

Электронная почта по составу элементов и принципу работы практически повторяет систему обычной (бумажной) почты, заимствуя как термины (почта, письмо, конверт, вложение, ящик, доставка и другие), так и характерные особенности — простоту использования, задержки передачи сообщений, достаточную надёжность и, в то же время, отсутствие гарантии доставки.

В данной работе рассматривается реализация приложения по приему и отправке почтовых сообщений. Функциональное назначение данного приложения – хранение информации о пользователях, прием и отправка сообщений.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

При разработке программных продуктов необходимо провести анализ существующих вариантов для решения задачи. В данном разделе рассматриваются общие теоретические вопросы, освещающие технологические аспекты, необходимые для реализации программного продукта. В частности, производится выбор СУБД, архитектуры приложения, языка программирования.

1.1 Анализ типов СУБД

Существуют два вида СУБД: реляционные (*SQL*) и нереляционные (*NoSQL*). Чтобы выбрать тот или иной вид СУБД, нужно знать характерные отличия их друг от друга.

Можно выделить следующие отличительные признаки:

- Разница в структуре и типе хранимых данных. В реляционных СУБД должна быть однозначно определена структура хранения данных, в то время как в *NoSQL* БД нет таких строгих ограничений, и они допускают хранение любого типа данных. Для системы приема отправки почтовых сообщений подходит структура данных с однозначно определенными связями между объектами таблиц.
- Различия в структуре запросов. В *SQL* БД данные могут быть получены только посредством языка *SQL*, так как реляционные СУБД соответствуют стандартам *SQL*. В нереляционных СУБД нет четкого определения, как должен выглядеть запрос и поэтому для каждой *NoSQL* БД реализуется свой способ работы с данными. Для системы приема отправки почтовых сообщений требуется построение множества запросов с выборками по условию, и использование нереляционной СУБД будет

сопряжено с проблемами с созданием запросов, поэтому выбор реляционной СУБД для работы здесь очевиден.

- Масштабируемость. Оба вида СУБД могут хорошо масштабироваться по вертикали, однако по горизонтали лучше растягиваются именно нереляционные СУБД. системы приема отправки почтовых сообщений больше подходит вариант с лучшим горизонтальным масштабированием.
- Надежность. *SQL* базы данных являются более надежными, по сравнению с *NoSQL* БД. Поэтому реляционные БД более предпочтительны для выбора в качестве решения для системы.
- Поддержка. Реляционные СУБД появились гораздо раньше *NoSQL*, поэтому существуют уже готовые платные и бесплатные решения и поддержка, и найти ответ на свой вопрос гораздо проще, чем, когда речь идет о сложном решении в нереляционной СУБД. И снова выбор в пользу *SQL*.
- Хранение и доступ к сложным структурам данных. *SQL* базы данных выигрывают в производительности по сравнению с нереляционными СУБД, так как изначально предполагалось, что они будут работать именно со сложными структурами данных. Для системы приема отправки почтовых сообщений снова больше подходит *SQL* БД.

Более предпочтительным является использование реляционной СУБД.

1.2 Выбор реляционной СУБД

Для выбора наиболее подходящей РСУБД необходимо проанализировать имеющиеся на рынке популярные и развивающиеся *SQL* базы данных. Таковыми являются: *MySQL*, *PostgreSQL*, *MS SQL Server*, *Oracle Database*.

1.2.1 MS SQL Server

Microsoft SQL Server – разработанная корпорацией майкрософт, система управления реляционными базами данных. Основным языком для написания запросов является *Transact-SQL*. Данная РСУБД подходит как для создания персональных баз данных, так и для реализации БД масштаба компании. Последняя версия продукта – *SQL* Server 2019. В данной версии появилась возможность отслеживать, как меняются данные с течением времени (*temporal data support* – временная поддержка данных). А также эта версия поддерживает динамическую маскировку данных (*dynamic data masking*), которая дает гарантию того, что только авторизованные пользователи могут получить доступ к конфиденциальным данным.

Достоинства РСУБД *MS SQL Server*:

- простота использования;
- возможность регулировки и отслеживания уровня производительности, чтобы уменьшить загрузку;
- последняя версия продукта работает стабильно;
- возможность интеграции с другими продуктами *Microsoft*;
- возможность визуализации на мобильных устройствах.

Недостатки РСУБД *MS SQL Server*:

- очень высокая стоимость продукта для юридических лиц;
- возможны проблемы в работе служб интеграции импорта файлов;
- высокая ресурсоемкость *SQL* Server (даже при настройке производительности).

При всех упомянутых достоинствах, у данной РСУБД есть очень существенный недостаток, который делает ее недоступной для использования – ее высокая цена. Бесплатная версия не обладает достаточными возможностями для полной реализации базы данных для системы приема отправки почтовых сообщений.

1.2.2 Oracle Database

Oracle Database – объектно-реляционная система управления базами данных, разработанная компанией *Oracle*. История данной ОРСУБД берет начало с конца 70-х годов прошлого века. В настоящее время доступна последняя версия продукта, которая вышла в 2019 году – 19c (суффикс «c» обозначает английское слово «*cloud*» – облако). Эта версия поддерживает подключение баз данных через облачные сервисы, что позволяет размещать базу данных на нескольких серверах и хранить огромный объем данных. Возможность использования подключаемых БД обеспечивает мультиарендность (обслуживание множества организаций-клиентов – «арендаторов») и живую миграцию баз данных. Улучшена безопасность проведения операций – каждая транзакция проводится изолированно от остальных.

Достоинства ОРСУБД *Oracle Database*:

- надежность;
- современный функционал и новые разработки, доступные в последней версии;

Недостатки ОРСУБД *Oracle Database*:

- высокая стоимость продукта для юридических лиц;
- для системы необходимы аппаратные ресурсы, обладающие достаточной мощностью и производительностью;

К сожалению, высокая стоимость не позволяет использовать данную ОРСУБД, бесплатная версия позволяет развертывать небольшие по объему базы данных.

1.2.3 MySQL

MySQL – свободная реляционная система управления базами данных. С 2010 года права на торговую марку получила компания *Oracle*. *MySQL* является бесплатным программным обеспечением, регулярно выходят новые версии, дополняющие функционал. Существуют и платные версии для коммерческих организаций. В бесплатной версии сделан упор на надежность и скорость работы, но не на полноту функционала. *MySQL* является довольно гибкой СУБД за счёт возможности использовать множество типов таблиц.

Достоинства *MySQL*:

- бесплатная для личного использования;
- имеет хорошую документацию;
- безопасность;
- возможность использования множества функций в бесплатной версии;
- возможность взаимодействия с другими базами данных, например *DB2* и *Oracle*.

Недостатки *MySQL*:

- проблемы с надежностью;
- платная для коммерческого использования
- некоторые ограничения функционала;
- нет встроенной поддержки *XML* и *OLAP*;
- для бесплатной версии доступна только платная поддержка.

Несмотря на кажущиеся преимущества, система в основном используется для мелких и средних проектов и не подходит для крупных, а также не бесплатная.

1.2.4 PostgreSQL

PostgreSQL – свободная объектно-реляционная система управления базами данных с открытым исходным кодом, разработчиком которой

является сообщество *PostgreSQL*. Базируется на языке *SQL*. Последняя доступная версия программы – 12. *PostgreSQL* активно разрабатывается сообществом, в него постоянно добавляются новые функции. Последнее время все больше используется в крупных коммерческих проектах.

Достоинства *PostgreSQL*:

- бесплатная и для личного и для коммерческого использования;
- хорошо масштабируется;
- возможность обрабатывать терабайты данных;
- поддержка формата *JSON*;
- существует множество готовых, встроенных функций;
- хорошая документация;
- большое сообщество с поддержкой.

Недостатки *PostgreSQL*:

- неэффективная репликация данных;
- трудности с обновлением.

PostgreSQL выигрывает по всем параметрам и самое главное – программа бесплатно распространяется. Поэтому было принято решение использовать именно эту ОРСУБД.

В проекте используется версия 12, на момент начала разработки она была выбрана как наиболее стабильная.

1.3 Выбор языка программирования

При реализации любого программного продукта одним из самых важных этапов является выбор языка программирования. От этого зависит продуктивность и чистота кода, а также длительность периода создания минимально жизнеспособного продукта.

В настоящее время наиболее популярными языками программирования для разработки веб-приложений являются: *Go*, *Java*, *JavaScript*, *Python*. Такие

языки как *C++, Php* и *Ruby* в последнее время теряют популярность в веб-разработке.

1.3.1 Язык программирования Go

Go (также известный как *Golang*) — компилируемый многопоточный язык программирования, разработанный компанией *Google*. Язык разрабатывался для проектирования высокопроизводительных программ и для решения проблем большинства языков. Язык проектировался с целью нивелировать такие проблемы при разработке как медленная сборка программы, сложность при разработке инструментария, проблемы межъязыкового взаимодействия и другие.

Но данный язык не является объекто-ориентированным в полном смысле этого понятия, так как в язык сознательно не была включена концепция наследования. Язык обладает меньшей переносимостью на разные платформы, так как компилируется в машинный код.

1.3.2 Язык программирования Python

Python – высокоуровневый язык программирования, который ориентирован на повышение производительности разработчика и читаемости кода. Этот язык очень гибкий и обладает широким функционалом, а код высокой степенью читаемости. Но у данного языка есть существенный недостаток – он обладает низким быстродействием и поэтому часто используется как язык второго плана.

1.3.3 Язык программирования JavaScript

JavaScript – мультипарадигменный язык программирования. Язык не принадлежит какой-либо организации или компании. Наиболее часто используется в качестве языка сценариев веб-страниц. *JavaScript* поддерживает императивный, декларативный и объектно-ориентированный стили программирования. *JavaScript* часто используется в таких программных продуктах, которые в свою очередь используются в веб-разработке. Так как язык является интерпретируемым, то

производительность и безопасность зависит от конкретной реализации интерпретатора («движка»).

1.3.4 Язык программирования Java

Java – сильно типизированный объектно-ориентированный язык программирования. В настоящее время разработкой занимается компания *Oracle*. Основной особенностью данного языка является то, что программы на *Java* транслируются в байт-код, который выполняется на виртуальной машине *Java*. Виртуальная машина *Java* в свою очередь является интерпретатором и передает инструкции оборудованию, что обеспечивает легкую переносимость программы на разные платформы. В настоящее время *Java* является одним из самых популярных языков программирования.

1.3.5 Выбор языка программирования

В качестве языка программирования был выбран язык *Java для backend разработки и JavaScript для frontend разработки*. В отличие от программ написанных на *Python* программы на *Java/JS* обладают большей производительностью, по сравнению с языком *Go Java/JS* давно используется программистами, имеет большое сообщество, хорошую документацию, множество популярных фреймворков. Для упрощения и ускорения процесса разработки был выбран фреймворк Spring. Данный фреймворк является универсальным и используется для создания бизнес-приложений.

1.4 Описание архитектуры REST API

REST – это архитектурный стиль взаимодействия компонентов распределенного веб-приложения. Принципы *REST* сформулировал Рой Филдинг в своей диссертации в 2000 году, но не существует официально принятого стандарта или спецификации *REST* (в отличие, например, от протокола *SOAP*). *API*-интерфейс может считаться *RESTful* только в том случае, если соблюдены все требования. При создании системы приема

отправки почтовых сообщений были учтены данные требования, и поэтому веб-приложение является *RESTful*.

Существует шесть обязательных ограничений для проектирования *REST*-приложений:

1.4.1 Отсутствие состояния

Взаимодействие между клиентом и сервером строится по следующему условию: в моменты времени между запросами никакая информация о клиенте не должна храниться на сервере, а каждый запрос от клиента должен быть составлен таким образом, чтобы на сервер поступила вся необходимая информация о транзакции для ее завершения. При этом информация о состоянии сессии сохраняется у клиента. В то время как происходит обработка запросов от клиента, считается, что он находится в переходном состоянии. Данное ограничение позволяет системе лучше масштабироваться, так как отсутствие необходимости хранить информацию о состояниях высвобождает ресурсы сервера, которые могут быть направлены на обработку одновременно большего количества клиентов.

1.4.2 Кэширование

Клиенты могут выполнять кэширование ответов сервера. В этом случае ответы сервера должны быть обозначены как кэшируемые, если поступила новая информация, или некэшируемые, если поступившие данные уже передавались ранее. Грамотно построенное кэширование способно уменьшить взаимодействие между клиентом и сервером, что увеличивает производительность и масштабируемость распределенной веб-системы в целом.

1.4.3 Единообразие интерфейса

Для эффективного взаимодействия компонентов и кэширования в сети необходим унифицированный интерфейс, позволяющий независимо развиваться отдельным сервисам.

В свою очередь для унифицированных интерфейсов существует четыре условия:

- Идентификация ресурсов. Любая информация может быть ресурсом – для этого она должна иметь имя. Каждый ресурс должен быть идентифицирован с помощью идентификатора, не меняющегося при изменении состояния ресурса. Идентификатором в *REST* является *URI*.
- Манипуляция ресурсами через представление. Ресурсы могут быть представлены различными способами, например: *JSON*, *HTML*, *XML* описание и так далее. Представление является описанием текущего состояния ресурса и используется для выполнения операций над ресурсом. Взаимодействия клиентов с ресурсами происходит посредством представлений.
- «Самоописываемые» сообщения. Каждый запрос (ответ) хранит в себе всю необходимую информацию, чтобы понять, как его нужно обработать. Для обработки одного запроса не должно быть дополнительных сообщений.
- Гипермедиа, как средство изменения состояния приложения. Для навигации по *API* должен быть использован гипертекст, что позволяет клиентам обнаруживать ресурсы посредством гиперссылок.

1.4.4 Многослойная система

В архитектуре *REST* возможно разделение системы на иерархию слоев, но с условием: отдельный компонент системы может видеть только компоненты следующего уровня. Использование промежуточных прокси-серверов позволяет увеличить масштабируемость, сбалансировав нагрузку и

распределив кэширование. Также это позволяет использовать политику безопасности для обеспечения конфиденциальности данных.

1.4.5 Код по требованию

Является необязательным требованием. В архитектуре *REST* данное условие разрешает загружать программы (код) для выполнения на стороне клиента.

1.5 Выбор фреймворка для Frontend разработки

Фронтендом называют клиентскую сторону пользовательского интерфейса по отношению к программно-аппаратной части сервиса.

Frontend — это разработка функциональности и пользовательского интерфейса, работающих на клиентской стороне приложения или веб-сайта. Сюда относят всё, что пользователь видит, открывая веб-страницу. Для создания удобного и востребованного продукта фронтенд-разработчики сотрудничают с программистами, дизайнерами, UX-аналитиками.

1.5.1 Фреймворк Angular

Angular — это среда JavaScript MVVM, основанная в 2009 году, которая отлично подходит для создания интерактивных веб-приложений.

Преимущества Angular:

- Angular используется вместе с Typescript. Он имеет исключительную поддержку для этого.
- Angular-language-service — обеспечивает интеллектуальные возможности и автозаполнение шаблона HTML-компонента.
- Новые функции, такие как generation Angular, использующие библиотеки npm из CLI, generation, и разработка компонентов, использующая Angular.

- Подробная документация, позволяющая разработчику получить всю необходимую информацию, не прибегая к помощи его коллег. Однако это требует больше времени для обучения.
- Односторонняя привязка данных, которая обеспечивает исключительное поведение приложения, что сводит к минимуму риск возможных ошибок.
- MVVM (Model-View-ViewModel), которая позволяет разработчикам работать отдельно над одним и тем же разделом приложения, используя один и тот же набор данных.
- Внедрение зависимостей от компонентов, связанных с модулями и модульностью в целом.
- Структура и архитектура, специально созданные для большой масштабируемости проекта.

Недостатки Angular:

- Разнообразие различных структур (Injectables, Components, Pipes, Modules и т. д.) усложняет изучение по сравнению с React и Vue.js, у которых есть только «Component».
- Относительно медленная производительность, учитывая различные показатели. С другой стороны, это можно легко решить, используя так называемый «ChangeDetectionStrategy», который помогает вручную контролировать процесс рендеринга компонентов.

Компании, использующие Angular: Microsoft, Autodesk, MacDonald’s, UPS, Cisco Solution Partner Program, AT&T, Apple, Adobe, GoPro, ProtonMail, Clarity Design System, Upwork, Freelancer, Udemy, YouTube, Paypal, Nike, Google, Telegram, Weather, iStockphoto, AWS, Crunchbase.

1.5.2 Фреймворк React

React — это библиотека JavaScript, разработанная Facebook в 2013 году, которая отлично подходит для создания современных одностраничных приложений любого размера и масштаба.

Преимущества React:

- Легко изучить, благодаря простому дизайну, использованию JSX (HTML-подобный синтаксис) для шаблонов и очень подробной документации. Разработчики тратят больше времени на написание современного JavaScript и меньше беспокоятся о коде, специфичном для фреймворка.
- Очень быстрая, благодаря реализации React Virtual DOM и различным оптимизациям рендеринга.
- Отличная поддержка рендеринга на стороне сервера, что делает его мощной платформой для контент-ориентированных приложений.
- Первоклассная поддержка Progressive Web App (PWA) благодаря генератору приложений `create-react-app`.
- Привязка данных является односторонней, что означает меньше нежелательных побочных эффектов.
- Redux, самая популярная платформа для управления состоянием приложений в React, ее легко учить и использовать.
- React реализует концепции функционального программирования (FP), создавая простой в тестировании и многократно используемый код.
- Приложения могут быть созданы с помощью TypeScript или Facebook’s Flow, имеющими встроенную поддержку JSX.
- Переход между версиями, как правило, очень прост: Facebook предоставляет «кодовые модули» для автоматизации большей части процесса.
- Навыки, полученные в React, могут быть применены к разработке на React Native.

Недостатки React:

- React не однозначен и оставляет разработчикам возможность выбирать лучший способ развития. Это может быть решено сильным лидерством проекта и хорошими процессами.
- Сообщество делится по способам написания CSS в React, которые разделяются на традиционные таблицы стилей (CSS Modules) и CSS-in-JS (т.е. Emotion и Styled Components).
- React отходит от компонентов на основе классов, что может стать препятствием для разработчиков, которым более комфортно работать с объектно-ориентированным программированием (ООП).
- Смешивание шаблонов с логикой (JSX) может сбить с толку некоторых разработчиков при первых знакомствах с React.

Компании, использующие React: Facebook, Instagram, Netflix, New York Times, Yahoo, Khan Academy, Whatsapp, Codecademy, Dropbox, Airbnb, Asana, Atlassian, Intercom, Microsoft, Slack, Storybook и многие другие

1.5.3 Фреймворк Vue.js

Vue.js — это JavaScript-фреймворк, основанный в 2013 году, который идеально подходит для создания высоко адаптируемых пользовательских интерфейсов и сложных одностраничных приложений.

Преимущества Vue.js:

- Усиленный HTML. Это означает, что Vue.js имеет много характеристик схожих с Angular, а это, благодаря использованию различных компонентов, помогает оптимизации HTML- блоков.
- Подробная документация. Vue.js имеет очень подробную документацию, которая может ускорить процесс обучения для

разработчиков и сэкономить много времени на разработку приложения, используя только базовые знания HTML и JavaScript.

- Адаптивность. Может быть осуществлен быстрый переход от других фреймворков к Vue.js из-за сходства с Angular и React с точки зрения дизайна и архитектуры.
- Потрясающая интеграция. Vue.js можно использовать как для создания одностраничных приложений, так и для более сложных веб-интерфейсов приложений. Важно, что небольшие интерактивные элементы можно легко интегрировать в существующую инфраструктуру без негативных последствий.
- Масштабирование. Vue.js может помочь в разработке довольно больших шаблонов многократного использования, которые могут быть сделаны почти за тоже время, что и более простые.
- Крошечный размер. Vue.js весит около 20 КБ, сохраняя при этом свою скорость и гибкость, что позволяет достичь гораздо лучшей производительности по сравнению с другими платформами.

Недостатки Vue.js:

- Недостаток ресурсов. Vue.js по-прежнему занимает довольно небольшую долю рынка по сравнению с React или Angular, что означает, что обмен знаниями в этой среде все еще находится на начальной стадии.
- Риск чрезмерной гибкости. Иногда у Vue.js могут возникнуть проблемы при интеграции в огромные проекты, и пока еще нет опыта возможных решений, но они обязательно появятся в ближайшее время.

Компании, которые используют Vue.js: Xiaomi, Alibaba, WizzAir, EuroNews, Grammarly, Gitlab и Laracasts, Adobe, Behance, Codeship, Reuters.

1.6 Выбор системы управления версиями Java

Главное отличие между Maven и Gradle является то, что Maven - это инструмент управления и понимания проектов программного обеспечения, который управляет сборками проектов, отчетами и документами, а Gradle - это инструмент автоматизации сборки с открытым исходным кодом, ориентированный на гибкость и производительность

1.6.1 Maven

Maven - это инструмент управления проектами, который управляет сборками проектов, зависимостями, распространением, выпусками и т. Д. Здесь сборка программного обеспечения относится к процессу, с помощью которого исходный код преобразуется в автономную форму, которую можно запустить на компьютере.

Кроме того, Maven позволяет компилировать, распространять, документировать и совместно работать в команде. Кроме того, репозиторий Maven - это каталог упакованного файла JAR с файлом pom.xml. Он имеет информацию о конфигурации для создания проекта. Здесь JAR - это пакет, который объединяет несколько файлов классов Java и связанных ресурсов в один файл для распространения.

1.6.2 Gradle

Gradle - это система автоматизации сборки с открытым исходным кодом. Он использует понятия Apache Ant и Apache Maven. Прежде всего, он был разработан для поддержки многопроектных сборок. Где, Gradle использует направленный ациклический график, чтобы определить порядок выполнения задач. Gradle стал популярным в течение короткого периода времени. Например, Google принял Gradle в качестве инструмента сборки по умолчанию для ОС Android.

Кроме того, Gradle основан на доменно-специфическом языке (DSL). Он не использует XML. Потому что он предназначен для решения конкретной проблемы. Кроме того, он поддерживает жизненный цикл программного обеспечения от компиляции до статистического анализа и тестирования до упаковки и развертывания.

Кроме того, Gradle предоставляет множество преимуществ. Это позволяет применять общие принципы проектирования к процессу сборки. Это помогает разработчикам создавать хорошо структурированные и поддерживаемые сборки для нескольких проектов. Кроме того, он предоставляет различные методы для управления зависимостями.

1.7 Выбор пакетного менеджера

Npm и Yarn – два самых популярных менеджеры пакетов для JavaScript. Менеджер пакетов является естественным способом автоматизации процесса установки, обновления и удаления сторонних модулей, которые хранятся в общей базе модулей. В этой статье будет рассказано в чем разница между двумя самыми популярными менеджерами пакетов для JavaScript – npm и Yarn.

Npm – это менеджер пакетов, входящий в состав Node.js. Он использует клиент командной строки и базу данных, состоящую из общедоступных и приватных пакетов, известной как npm registry. Пользователи могут получить доступ к базе через сайт или через консоль.

Yarn был разработан в Facebook чтобы избавится от недостатков npm. Технически Yarn не является заменой npm, так он берет информацию про модули из базы npm. По сути Yarn это новый установщик который по прежнему базируется на структуре заданной npm. В Yarn доступны все те же пакеты что и в npm, поэтому, переезд с npm на Yarn не требует больших усилий.

Управления версиями в файле ‘package.json’ иногда становится беспорядочным. Файл yarn.lock помогает упорядочить эту путаницу. При добавлении новой зависимости, Yarn обновляет yarn.lock файл, принцип работы похож на Gemfile.lock в Ruby. yarn.lock файл гарантирует что на каждом устройстве будет установлена одинаковая версия пакета. lock файлы так называются, потому что они фиксируют версии зависимостей во время их установки. lockfile состоит из отсортированных ключей чтобы гарантировать минимальный изменения в файловой структуре папки node_modules.

В более ранних версиях npm, такая функциональность была реализована с помощью команды упаковывания(npm shrinkwrap). Тем не менее, shrinkwrap файл не генерировался автоматически, и требовал постоянной поддержки и обновления. Чтобы решить эту проблему npm ввел package-lock.json

Yarn на порядок быстрее всех версий npm меньше 5.0. Когда команда npm выпустила 5 версию они заявили что она будет в пять раз быстрее своих предшественников. Но все же npm остался медленнее чем Yarn

Главная проблема npm, это то что он автоматически запускает код зависимостей и позволяет добавлять зависимости на лету, хотя такие особенности дают много преимуществ, они также создаю уязвимости в безопасности. Так как Yarn устанавливает зависимости только с файлов yarn.lock или package.json он считается более безопасным. Также Yarn проверяет контрольные суммы перед установкой чтобы гарантировать целостность каждого пакета.

Yarn будет все более популярными благодаря своей хорошей производительности, и многочисленным полезным командам, для разработки был выбран именно он.

1.8 Описание серверного механизма Thymeleaf.

Thymeleaf — современный серверный механизм Java-шаблонов для веб- и автономных сред, способный обрабатывать HTML, XML, JavaScript, CSS и даже простой текст.

Основной целью Thymeleaf является создание элегантного и удобного способа шаблонизации. Чтобы достичь этого, Thymeleaf основывается на концепции Natural Templates, чтобы внедрить свою логику в файлы шаблонов таким образом, чтобы этот шаблон не влиял на отображения прототипа дизайна.

Thymeleaf также был разработан с самого начала с учетом стандартов Web, особенно HTML5, что позволяет вам создавать полностью соответствующие стандарту шаблоны.

Из коробки, Thymeleaf позволяет работать с шестью типами шаблонов, каждый из которых называется «Формат шаблона/Template Mode»:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

Итого: два markup фармата (HTML and XML), три текстовых формата (TEXT, JAVASCRIPT и CSS) и простой формат (RAW).

HTML формат подразумевает HTML стандарт, включая HTML5, HTML 4 и XHTML. Валидация не выполняется, и код/структура оригинального шаблона будут соблюдаться в максимально возможной степени.

Формат шаблона XML позволит создать XML шаблон. В этом случае ожидается, что код будет хорошо сформирован, иначе без закрытых тегов, без кавычек атрибутов и прочего парсер сгенерирует исключения. Обратите внимание, что не будет проведена проверка DTD или XML-схемы.

Режим шаблона TEXT позволяет использовать специальный синтаксис для шаблонов без разметки. Примерами таких шаблонов могут быть текстовые письма или шаблонные документы. Обратите внимание, что HTML или XML-шаблоны также могут обрабатываться как TEXT, и в этом случае разметка не анализируется, и каждый тег, DOCTYPE, комментарий и прочее будет рассматриваться как простой текст.

Режим шаблона JAVASCRIPT позволит обрабатывать файлы JavaScript в приложении Thymeleaf. Это означает возможность использовать данные моделей внутри файлов JavaScript так же, как это можно сделать в HTML-файлах, но с интеграциями, специфичными для JavaScript, такими как специализированное экранирование или скриптинг. Режим шаблона JAVASCRIPT считается текстовым режимом и поэтому использует тот же специальный синтаксис, что и режим шаблона TEXT.

Режим шаблона CSS позволяет обрабатывать файлы CSS, участвующие в приложении Thymeleaf. Подобно режиму JAVASCRIPT, режим шаблона CSS также является текстовым режимом и использует специальный синтаксис обработки из режима шаблона TEXT.

Режим шаблона RAW просто не будет обрабатывать шаблоны. Он предназначен для вставки нетронутых ресурсов (файлов, ответов на URL и

тп.) в обрабатываемые шаблоны. Например, внешние, неконтролируемые ресурсы в формате HTML могут быть включены в шаблоны приложений, надежно зная, что любой код Thymeleaf, который эти ресурсы включает, не будет выполнен.

1.9 Описание фреймворка Spring

Spring Framework представляет собой просто контейнер внедрения зависимостей, с несколькими удобными слоями (например: доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC, веб-инфраструктура MVC). Это все позволяет вам быстрее и удобнее создавать Java-приложения.

1.9.1 Структура Spring

Spring — это не один какой-то конкретный фреймворк. Это скорее общее названия для целого ряда небольших фреймворков, каждый из которых выполняет свою работу.

Как видно, у спринга модульная структура. Это позволяет подключать только те модули, что нам нужны для нашего приложения и не подключать те, которыми мы заведомо не будем пользоваться. Насколько мне известно, то именно этот подход и помог спрингу обойти своего конкурента в то время (EJB) и захватить лидерство. Потому что приложения, использующие EJB тянули очень много зависимостей за собой, да и вообще получались медленные и неповоротливые.

1.9.2 Аутентификация в Spring Security

Аутентификация – это проверка, что пользователь есть тот, за кого себя выдает. Чтобы выполнить проверку, надо:

– Извлечь имя и пароль из HTTP-запроса. За это отвечает UsernamePasswordAuthenticationFilter

– Сравнить их с реальными именем и паролем, хранящимся в базе, на LDAP-сервере, во временной памяти приложения. Это делает AuthencationManager в методе authenticate(). Вызывается authenticate() из фильтра UsernamePasswordAuthenticationFilter сразу после извлечения имени/пароля из HTTP-запроса.

1.10 Выводы

Для реализации приложения была выбирана СУБД *PostgreSQL,* так как она предоставляет возможность использовать свой интерфейс и поддерживает работу с json, систему сборки Gradle т.к. она проста в изучении и управлении, пакетный менеджер Yarn за быстроту использования, язык Java с использованием фреймворка Spring для backend за огромное количество возможностей для разработки и с большим объемом документации, аутентификация Spring Security для проверки пользователей, JavaScript с фреймворком Vue, за большой потенциал в будущем и подробной документацией, с простым в освоениии синтаксисом.

2 РЕАЛИЗАЦИЯ

В данной работе приведены результаты разработки системы приема отправки почтовых сообщений (Рисунок 1). Функциональное назначение данной системы – прием и отправка сообщений. Для функционирования этой программы был создан сервер и клиентское приложение в виде страницы в браузере.

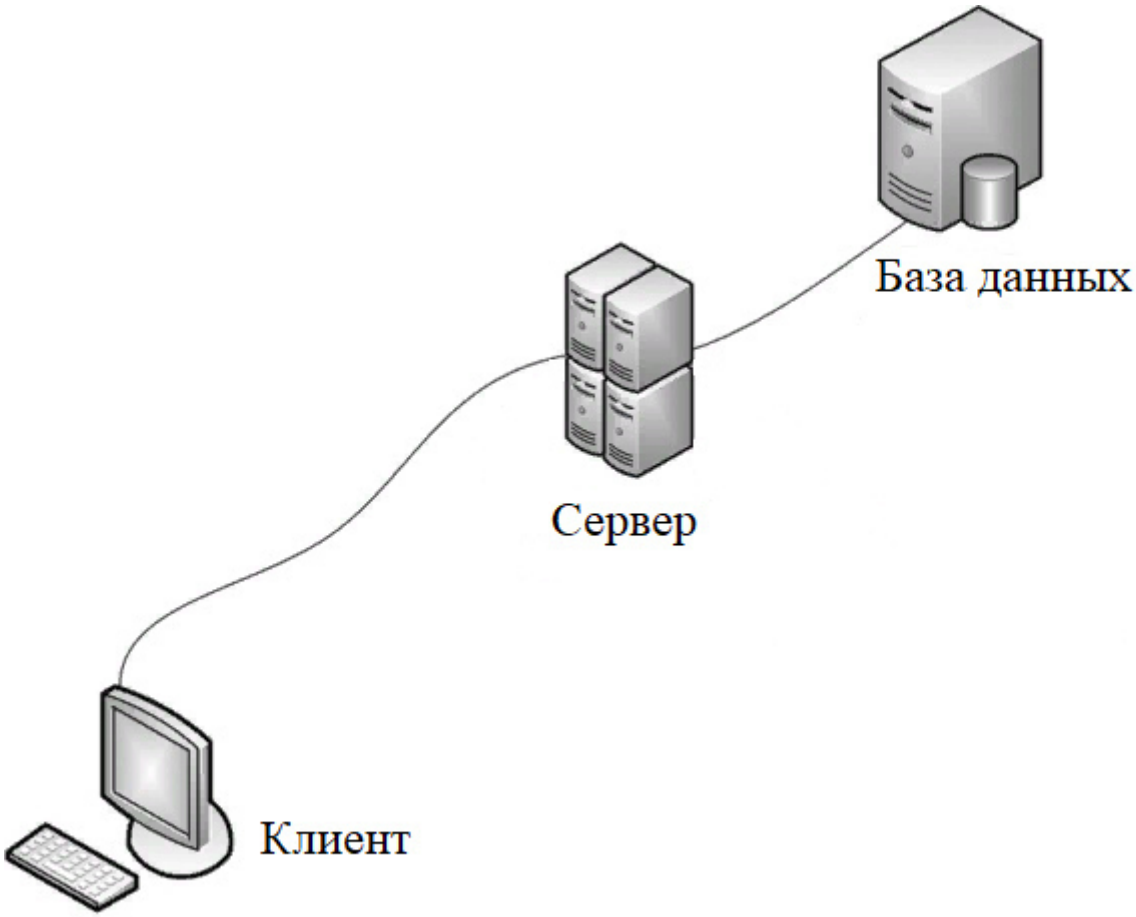


Рисунок 1 — Структурная схема

Процесс разработки этого компонента можно разделить на следующие этапы:

– разработка REST API, выполняющего обработку Post запросов для отправления почты и регистрации и Get запросы для отображения страницы в браузере с сообщениями из почтового ящика;

– разработка взаимодействия с базой данных для регистрации и авторизации пользователя

– разработка правил для авторизации пользователя

– создание страницы в браузере с возможностями общения с сервером

– разработка интерфейса страницы в браузере

2.1 Описание REST API

Разработка *API* один из важнейших этапов создания системы, так как от полноты реализации функционала *API* зависит удобство реализации остальных частей систем. *API*, описанное в данной выпускной работе, соответствует архитектурному стилю *REST* и поддерживает такую функцию как добавление пользователя.

Главный и единственный GetMapping(“/”) находится в классе **MainController.** Все GET запросы, получаемые на наш сервер в независимости от путей (кроме “/login” и “logout”), будут проходить через него. Это нужно для реализации нашего клиентского приложения, которое имеет только одну страницу и множество состояний(pages) в зависимости от роутинга.

Класс WebMvcConfig(Листинг 1) отвечает за перенаправление любых Get запросов на route “/”.

Листинг 1. Функция перенаправления на route “/”:

```
public WebServerFactoryCustomizer<ConfigurableWebServerFactory>
webServerCustomizer(){
    return container -> {
        container.addErrorPages(new ErrorPage(HttpStatus.NOT_FOUND, "/"));
        container.addErrorPages(new ErrorPage(HttpStatus.METHOD_NOT_ALLOWED,
"/"));
     };
}
```

Так же MainController не только возвращает html страницу, но и передает

вместе с ней данные в зависимости авторизован пользователь или нет (Листинг 2). Идет проверка есть ли авторизованный пользователь. Если нет от profile: false и возвращает стрницу. Если авторизован, то вместе со страницей пользователь клиент получит данные самого пользователя(email), сообщения с почтового ящика в папке “Inbox” и “Send”.

Листинг 2. Функция Get запроса на “/”:

```
    @GetMapping
        public String main(Model model) {
        HashMap<Object, Object> data = new HashMap<>();
        try{
            MyUserPrincipal userPrincipal = (MyUserPrincipal)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();

            data.put("profile", userPrincipal.getUsername());
            data.put("outbox",   findEmail("Sent",
userPrincipal.getUsername(), userPrincipal.getPassword()));
            data.put("inbox", findEmail("INBOX", userPrincipal.getUsername(),
userPrincipal.getPassword()));


        }catch (Exception e){
            data.put("profile", false);
        }

        model.addAttribute("frontendData", data);


        return "index";
    }
```

Класс **RegistrationController** обрабатывает Post запросы на “/registration”. Принимает Name, email, password, Предназначен для создания сущности в БД, В случае если пользователь не зарегистрирован, то проверяет, существует ли почтовый ящик с данными параметрами в сети (Листинг 3). Если существует, то добавляет пользователя в БД и возвращает 1, иначе возвращает -1.

Листинг 3. Функция проверки существования ящика:

```
private boolean findEmail(String email, String password) {
        Properties properties = new Properties();
        Folder inbox = null;
        Store store = null;
```

```
        try{
            properties.setProperty("mail.imap.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
            Session session = Session.getDefaultInstance(properties);
            store = session.getStore("imap");
            store.connect("imap." + email.split("@")[1], 993, email,
password);
            inbox = store.getFolder("INBOX");
        } catch (Exception e) {
            return false;
      } finally {
            if (store != null) {
                try {
                    store.close();
                } catch (MessagingException e) {
                    return false;
                    //e.printStackTrace();
                }
            }
        }
        return true;
    }
```

Класс **EmailController** обрабатывает Post запросы на “/send-email”. Принимает toSend, subject, text. Предназначен для отправки сообщения. Если письмо отправлено возвращает 1, иначе -1(Листинг 4).

Листинг 4. Функция отправки сообщения:

```
@PostMapping
public int sendEmail (@RequestBody Email email){
    Properties properties = new Properties();
    MyUserPrincipal userPrincipal = (MyUserPrincipal)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();

    properties.put("mail.smtp.host", "smtp." + email.getToSend().split("@")
[1]);
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.socketFactory.port", "465");
    properties.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
    Session session = Session.getDefaultInstance(properties,
            //Аутентификатор - объект, который передает логин и пароль
            new Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication()
{
```

```
                    return new
PasswordAuthentication(userPrincipal.getUsername(),
userPrincipal.getPassword());
                }
            });
    Message = new MimeMessage(session);
    try {
        message.setFrom(new InternetAddress(userPrincipal.getUsername()));

        message.setRecipient(Message.RecipientType.TO, new
InternetAddress(email.getToSend()));

        //Тема письма
        message.setSubject(email.getSubject());
        //Текст письма
        message.setText(email.getText());
    } catch (MessagingException e) {

    }
    try {
        Transport.send(message);
    } catch (MessagingException e) {
        e.printStackTrace();
    }

    return 1;
}
```

2.2 Структура БД

При разработке системы была использована СУБД *PostgreSQL*. Далее представлено описание структуры, разработанной БД. Таблица 1 содержит описание структуры таблицы «usr».

Таблица 1 — Структура таблицы «usr».

Название поля	**Описание поля**	**Тип данных**
id	Идентификатор	Long
name	Логин пользователя	String
Email	Email пользователя	String
password	Пароль пользователя от почты	String
registration	Время регистрации	Date

Настройка подключения к БД. спользуется spring jpa. Для соединения с бд нужно использовать файл application.properties. (Листинг 5) url-адрес нашей БД, username – пользователь БД. Password – пароль пользователя, generate-ddl – исользуются для автоматического создания таблицы в бд, используя сущность.

Листинг 5. Функция отправки сообщения:

```
spring.datasource.url=jdbc:postgresql://localhost:5000/kozhayevEMail
spring.datasource.username=postgres
spring.datasource.password=123
spring.jpa.generate-ddl=true

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

Сущность **User.java** имеет 6 полей: id, email, password, name. userpic, registration. Для связи сущности с БД используется класс UserDatalisRepo, с расширенной возможностью поиска по Email.

Листинг 6. Интерфейс взаимодействия с БД с сущностью User:

```
public interface UserDetailsRepo extends JpaRepository<User, Long> {
    User findByEmail(String email);
}
```

2.3 Реализация авторизации пользователя.

Для авторизации пользователя используется Spring security. В первую очередь настраиваются пути, на которые можно зайти неавторизованному пользователю, а на какие нельзя. Для этого используется Configuration класс **WebSecurityConfig**. Для используются следующие настройки: неавторизованным пользователям можно войти на “/”, “/registration”, “/welcome”, “/login”, “/js”, ”/error”, так же мы используем стандартную форму Sing in, предоставляемую и обслуживаемую полностью sprig’om и “/logout” (Листинг 7).

Листинг 7. Настройка Spring Security:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .antMatcher("/**")
            .authorizeRequests()
            .antMatchers("/", "/login**", "/registration**",  "/welcome**",
"/js/**", "/error**").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin().permitAll()
            .and().logout().logoutSuccessUrl("/").permitAll()
            .and()
```

```
            .csrf().disable();
}
```

Для того, чтобы spring security мог проверять существования пользователя в БД, нужно добавить еще 1 класс типа Service **MyUserDatailsService**, который использует ранее созданный нами интерфейс **UserDatailsRepo** для взаимодействий с БД.

Листинг 8. Настройка MyUserDatailsService:

```
@Autowired
private UserDetailsRepo userRepo;

@Override
public UserDetails loadUserByUsername(String username) {
    User user = userRepo.findByEmail(username);
    if (user == null) {
        throw new UsernameNotFoundException(username);
    }
    return new MyUserPrincipal(user);
}
```

2.4 Реализация клиентской части программы.

Для реализации страницы браузера используем html, js, Vue фреймвор с Material Design Vuetify. В файле index.html, который возвращает GetMapping”/”, находится только 1 div с id=”app”. В этот подставляется в зависимости от условий pages Vue. За рендер отвечает скрип main.js (Листинг 9), который был загружен, используя webpack, для облегчения разработки. В нем используется 3 зависимости. Vuetify для Material Design, VueResurce для отправлений запросов на сервер и принятие ответов, VueRouter для роутинга страниц (отображение содержимого странице в зависимости от router). Vue рекомендует делить все содержимое страницы на компоненты, затем использовать их в нужных местах.

Vue компонент состоит из 3 разделов: template, script, style. Template предназначен для html кода, разметки страницы, в случае с vue, разметка определенного компонента. Script используется для объявления переменных,

создание методов, объявления и импорта более мелких компонентов, props для получение данных от других компонентов. Все Vue системы можно описать деревом, где в одном компоненте (app.vue в этом случае), вызываются более мелкие компоненты и так далее.

Листинг 9. Main.js:

```
Vue.use(Vuetify)
Vue.use(VueResource)
Vue.use(VueRouter)


new Vue({
    el: '#app',
    vuetify: new Vuetify(),
    router: new VueRouter({
        mode: 'history',
    }),
    render: a => a(App)
})
```

Файл App.vue самый главные компонент, он и рендерится в index.html в .

Листинг 10. Template app.vue:

```
<template>
    <v-app>
        <Welcome v-if="!profile && this.$route.path ===
'/welcome'"></Welcome>
        <Menu v-if="this.$route.path === '/' && profile"></Menu>
        <div v-else-if="this.$route.path === '/' ">
            <meta http-equiv="refresh" content="1;
url=http://localhost:8080/welcome">
        </div>
        <Registration v-if="this.$route.path ===
'/registration'"></Registration>
    </v-app>
</template>
```

2.5 Взаимодействие классов

В системе по отправки и приему сообщений для взаимодействия ранее описанных СУБД и *REST API* архитектура разделена на основные функциональные блоки:

- классы-котроллеры – выполняют запросы *REST API*;
- классы-сущности – отвечают за описание сущностей БД в программе. Классы-сущности облегчают работу с данными, полученными, как результат использования функций классов-репозиториев;
- классы-репозитории – выполняют взаимодействие СУБД с сущностями. Данные передаются в эти классы из классов-сервисов;
- классы-сервисы – обеспечивают взаимодействие классов-контроллеров с классами-репозиториями посредством подготовки данных.
- классы-конфигураторы – настраивают определенные зависимости в проекте

UML диаграмма классов серверной части представлена на рисунке 2.

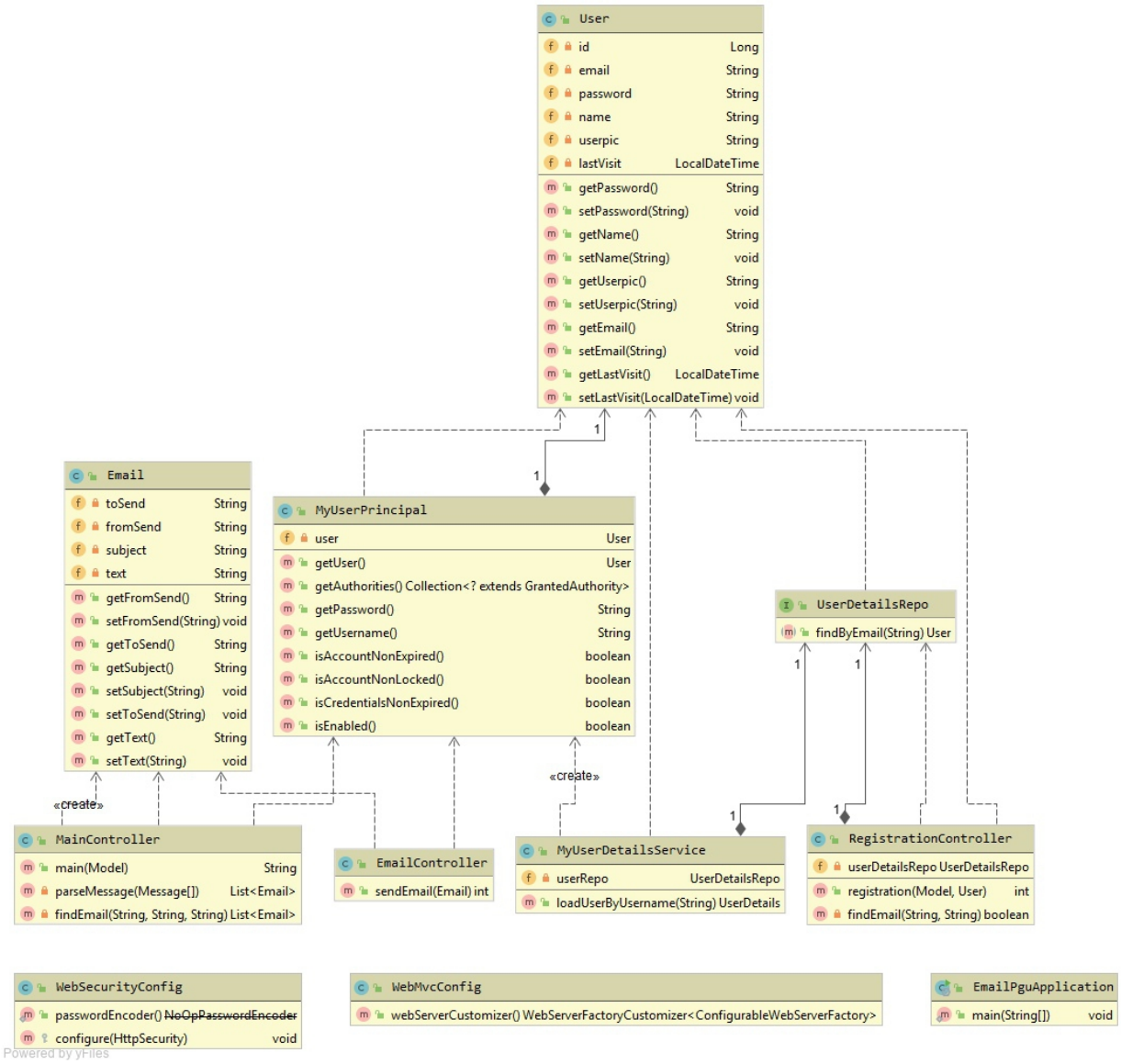


Рисунок 2 — *UML* диаграмма классов сервера

UML диаграмма, показывающая структуру frontend, представлена на рисунке 3.

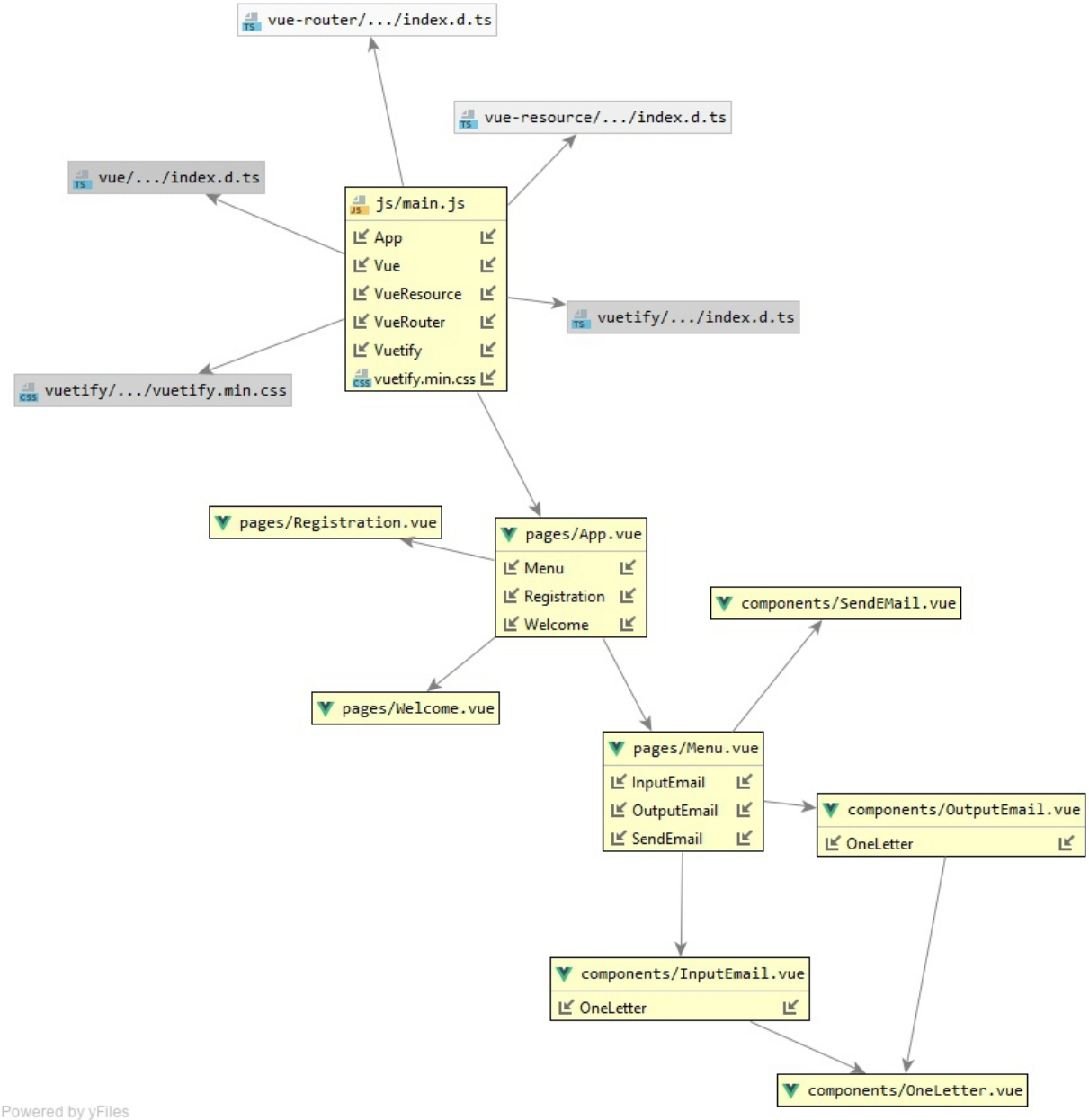


Рисунок 3 — *UML* диаграмма frontend

2.6 Программная реализация запроса к БД

Как упоминалось в предыдущем разделе, для реализации данного приложения используется язык программирования *Java* и фреймворк *Spring*. Был реализован *API* для работы с пользователями. На *UML* диаграмме (Рисунок 4) показана последовательность взаимодействия классов при обработке запроса «Получение пользователя по Email».

Рисунок 4 — Последовательность взаимодействия классов при обработке запроса «Получение пользователя по Email»

Пользователь использует функционал клиентского приложения (нажимает на кнопку), который в свою очередь генерирует запрос к клиентскому приложению. С клиентского приложения поступает запрос через Rest API на проверку существует ли данный клиент в системе. Класс-контроллер осуществляет проверку запроса на правильность, и если запрос удовлетворяет требованиям валидации, то класс-контроллер передает провалидированный запрос в класс-сервис. Класс-сервис подготавливает запрос к взаимодействию с объектами класса-репозитория. Класс-репозиторий отправляет запрос к БД, осуществляя взаимодействие с ней. Затем, когда необходимые данные получены из БД, класс-репозиторий преобразует данные в объекты программы и передает их в класс-сервис. Класс-сервис формирует ответ для класса-контроллера, класс-контроллер выдает результат, который передается клиентскому приложению.

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Разработанный веб-клиент представляет собой веб-приложение, в котором есть несколько логических страниц, для разделения функционала приложения.

При первом посещения основной страницы с путем “/”, если вы не авторизованный пользователь, вас перенаправит на страницу “/welcome” (Рисунок 5), на ней у вас будет выбор зарегистрировать, если вы первый раз на сайте, или войти, если вы уже являлись нашим пользователем.

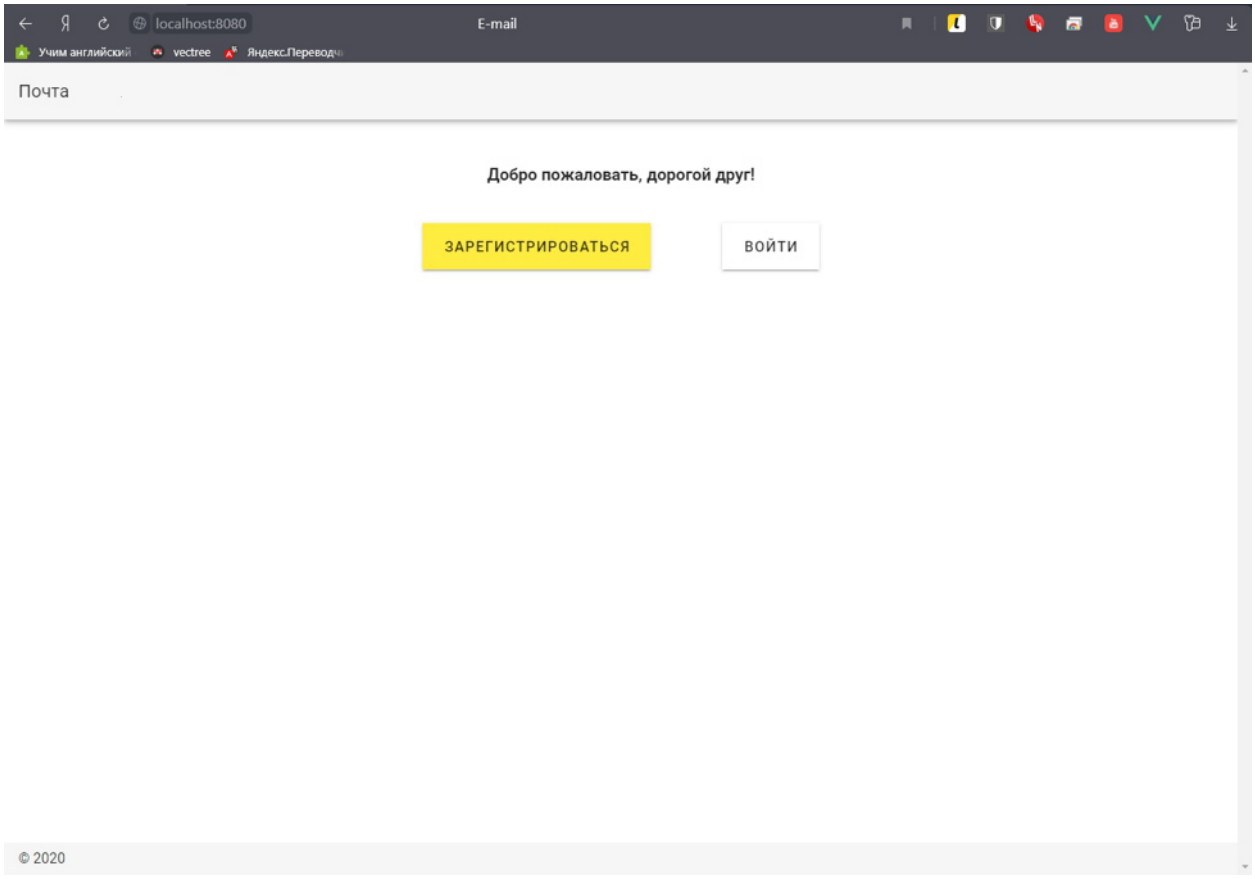


Рисунок 5 — Страница приветствия

При выборе пункта «Регистрация», вы будете перенаправленные на страницу регистрации(«/registration») (Рисунок 6). Здесь вам нужно заполнить поля (Email и пароль обязателен). Данные будут отправленные на сервер для проверки, существует ли доступ к почте по данном email и паролю. Если данный email уже зарегистрирован, выдаст соответствующее

сообщение и вы будете перенаправлены на страницу (“/login”). Если же данные не пройдут проверку, выдаст соответствующую ошибку и будет предложено повторить попытку регистрации.

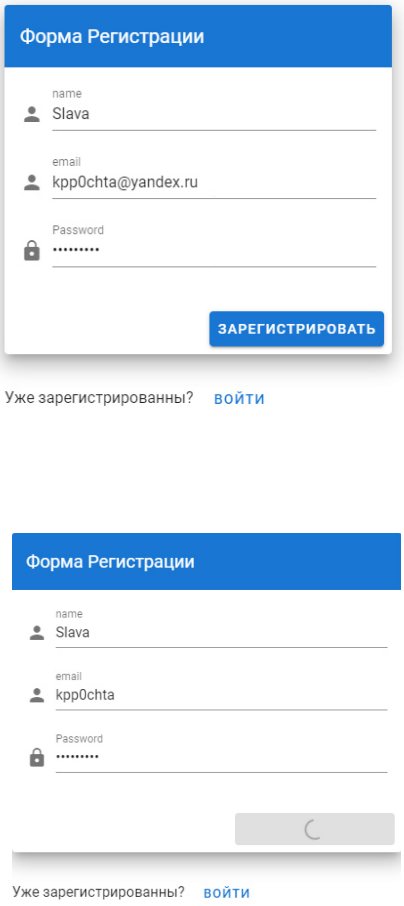


Рисунок 6 — Страница регистрации

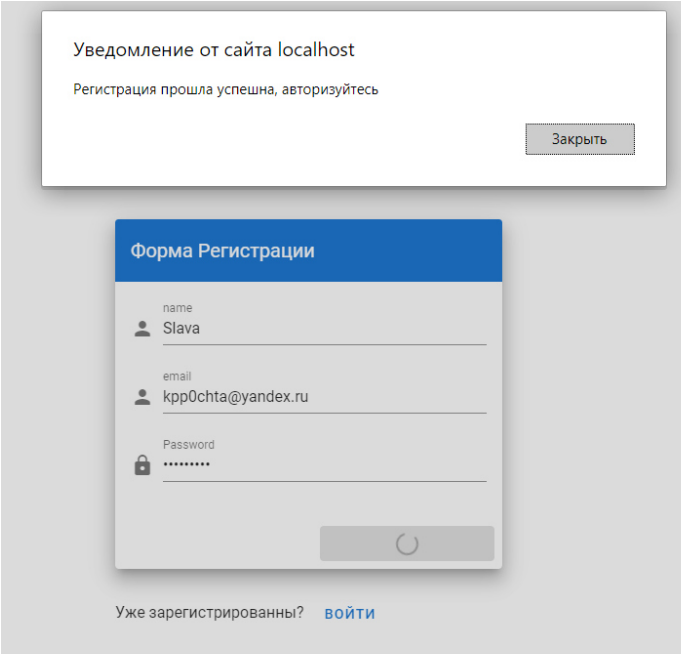


Рисунок 7 — Успешная регистрация

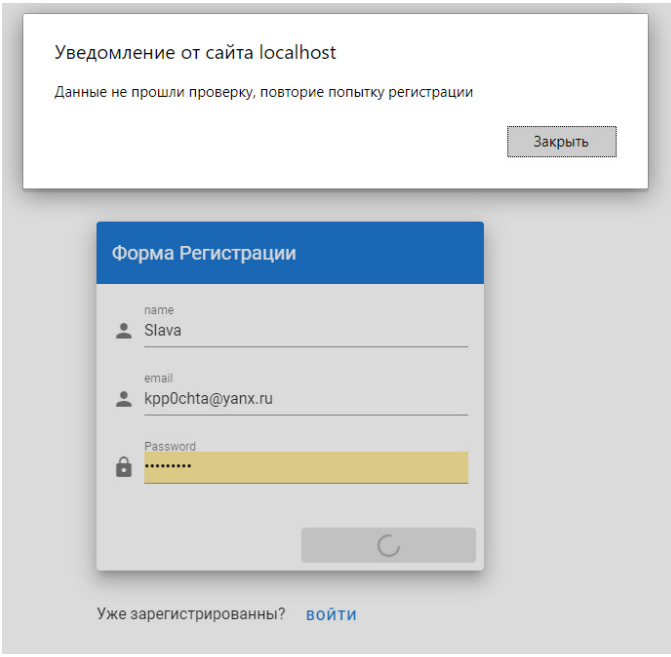


Рисунок 8 — Ошибка при регистрации

При выборе на странице Welcome или Registration пункт «Войти», вы попадаете на ссылку “/login” (Рисунок 9). Здесь нужно ввести данные, которые вводили при регистрации (e-mail и пароль). При некорректных данных выдаст соответствующую ошибку и по просит повторить попытку.

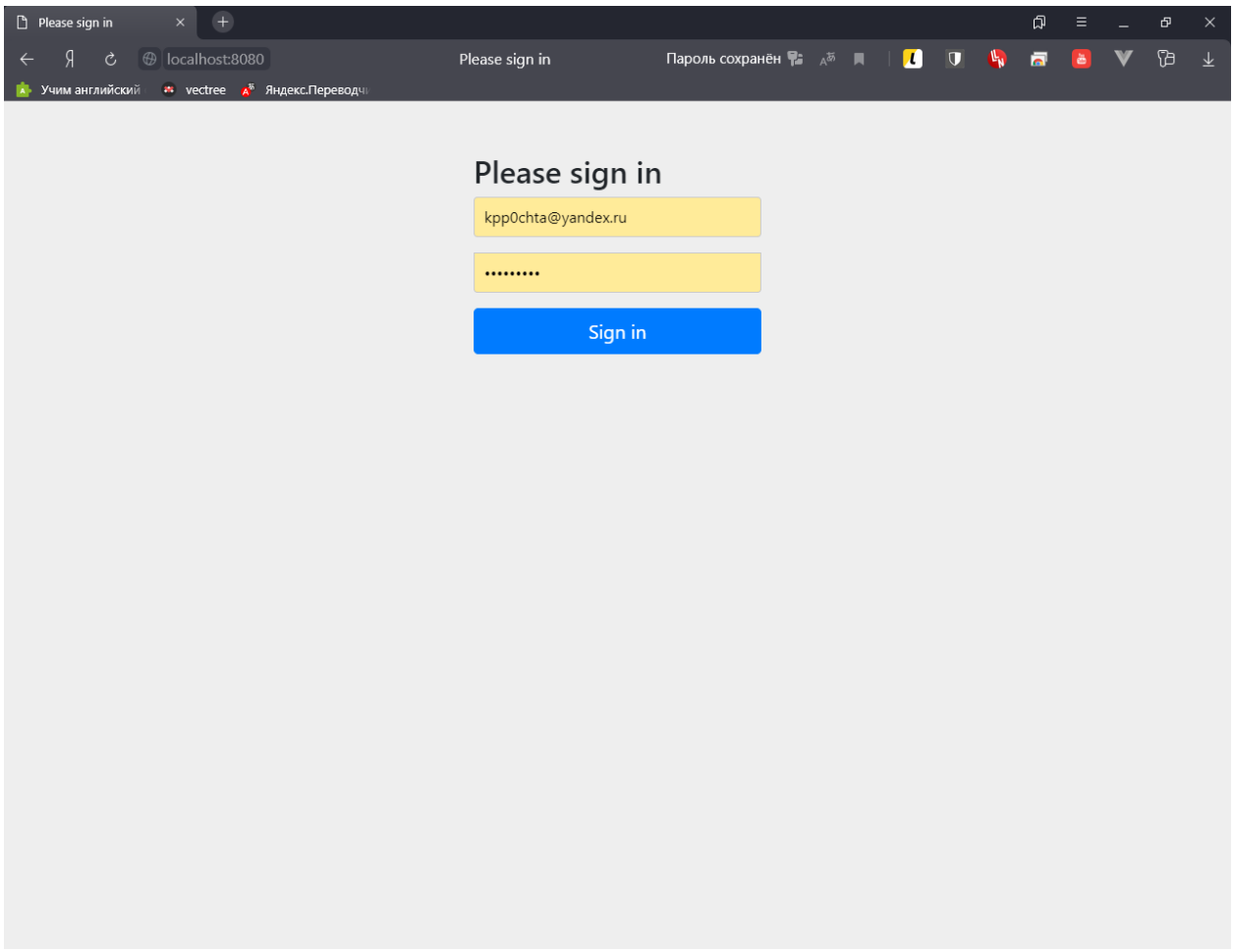


Рисунок 9 — Страница авторизации

В случае успешного выхода, вы будете перенаправленные на главную страницу приложения («/»). Здесь у вас предоставляется доступ к вашей почте и несколько логических блоков.

Первая возможно это открытие меню слева с доступом к написанию сообщения, просмотру входящих и отправленных вами сообщений.

Вторая возможность выхода из авторизованного аккаунта по нажатию ссылки выйти. В этом случае вы перенаправитесь на страницу “/logout”, а затем вновь на страницу “/welcome”.

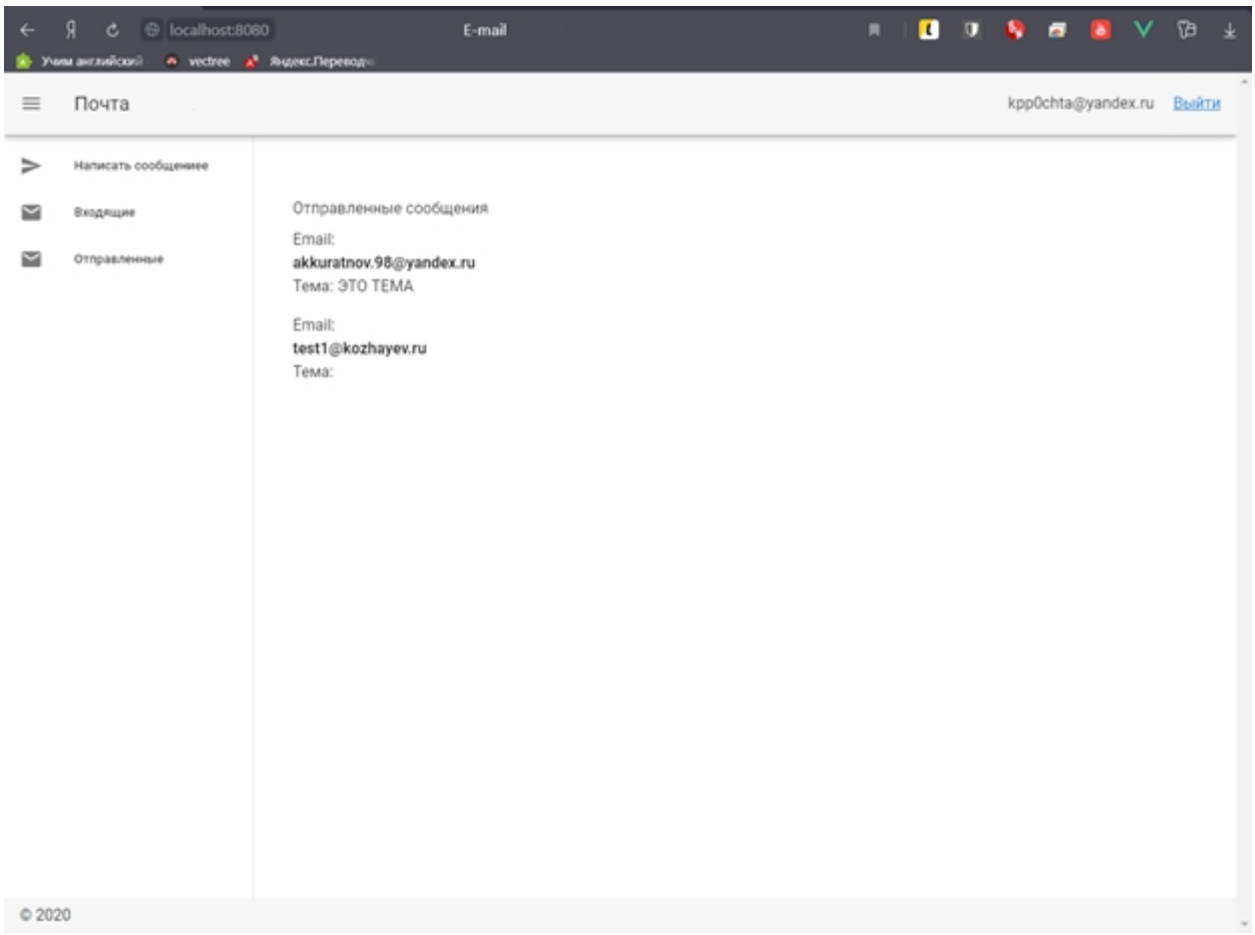


Рисунок 10 — Страница отправленных сообщений

Выбор в меню слева пункт «Написать сообщение». Откроется форма для отправки сообщения (Рисунок 11). Обязательные пункт «Кому отправить» и «Сообщение». Тема является не обязательным атрибутом.

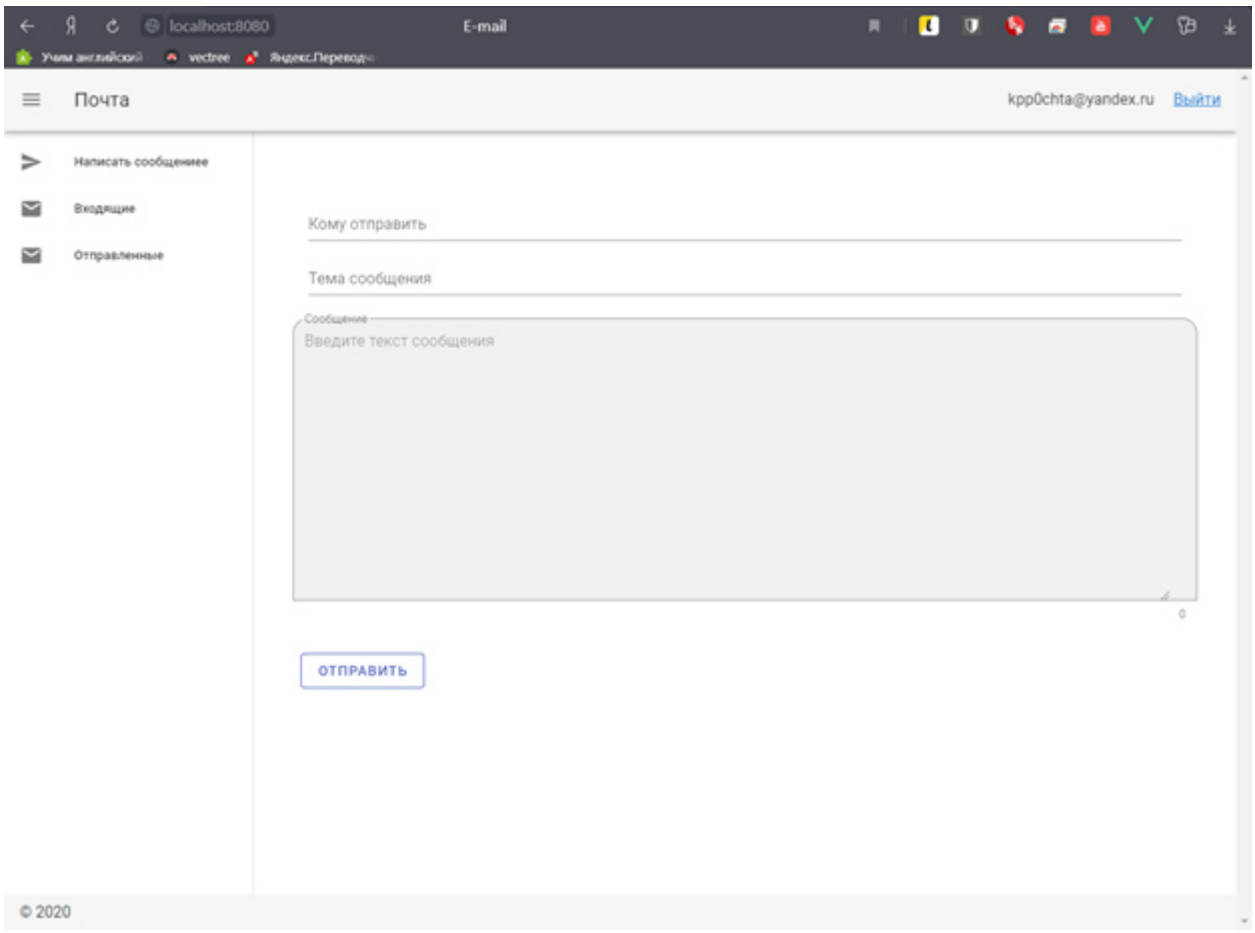


Рисунок 11 — Форма отправки сообщения

4 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

Воздействие персональных компьютеров на здоровье человека является важной проблемой современного общества. Во избежание вредного влияния необходимо соблюдать надлежащие меры безопасности, правильно планировать режим работы и выбирать рабочее место.

Искусственное освещение, используемое в помещениях, где сотрудники непосредственно связаны с персональными электронно-вычислительными машинами, обязано обеспечивать правильную работу глаз, а также приближать к оптимальным условиям зрительное восприятие, какое бывает при естественном освещении.

Для этого выделяют следующие требования к освещению на рабочих местах, оборудованных ПЭВМ:

– освещенность рабочего места должна быть порядка 300 – 500 лк. Запрещается работать в темном или полутёмном пространстве, а также в помещении с избыточным количеством света;

– искусственное освещение в помещениях для эксплуатации ПЭВМ должно быть распределено равномерно;

– естественный свет должен падать преимущественно слева от рабочего стола. Нельзя располагать оборудованное дисплеем рабочее место таким образом, чтобы в поле зрения оператора не попадали окна или осветительные приборы, а также они не должны находиться непосредственно за спиной пользователя;

– яркость бликов на экране ПЭВМ не должна превышать 40 кд / м2 и яркость потолка не должна превышать 200 кд / м2.

В качестве источников общего искусственного освещения лучше использовать люминесцентные лампы типа ЛБ с рассеивателями или экранирующими решетками, поскольку они создают равномерную освещенность путем рассеянного или отраженного распределения света и

исключают блики на экране монитора и клавиатуре. Соотношение яркости экрана и непосредственно ближайшего окружения не должно превышать 3:1.

Кроме того, на рабочие места необходимо устанавливать локальные источники света – настольные лампы с регулируемой яркостью и изменением расположения плафона.

Эти требования необходимо учитывать, поскольку без правильного размещения источников света на рабочих местах зрение сотрудников может падать, что приводит к хронической переутомляемости и усталости, а также утрате заинтересованности в результатах работы [10].

Помимо условий относительно освещенности помещения существуют правила организации рабочего стола:

– расстояние между рабочими столами с видеомониторами должно быть не мене 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м;

– экран видеомонитора должен находиться от глаз пользователя на расстоянии 600 - 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов;

– конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работ;

– рабочее кресло должно быть подъемно-поворотным, регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию;

– сиденье и спинка стула должны быть покрыты не электризующимися полумягкими материалами.

Кроме того, применение дисплеев с антибликовым, антирадиационным покрытием или защитных экранов поможет снизить вредное воздействие на здоровье человека при использовании персонального компьютера.

Для комфортной работы с персональным компьютером необходимо обеспечить оптимальные значения температуры воздуха в помещении (19-23 градуса Цельсия), а также относительную влажность воздуха (55%). Помещение, где находятся компьютеры, должно быть достаточно просторным и хорошо проветриваемым.

В связи с напряженностью трудового процесса у пользователей ПЭВМ рекомендуется организовать рабочую смену путем чередования работ с использованием ПЭВМ и без него. Продолжительность непрерывной работы не должна превышать 1 часа, а продолжительность перерыва – от 10 до 15 минут. При работе с ПЭВМ в ночную смену (с 22 до 6 ч.) продолжительность регламентированных перерывов следует увеличить на 30%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была изучена методология проектирования приложения и приобретены навыки работы с фрейворком *Spring и Vue*. При реализации поставленной задачи были получены следующие результаты:

- разработана структура базы данных для СУБД *PostgreSQL;*
- разработан прикладной интерфейс (*REST API*);
- разработана и отлажена система по приему и отправке почтовых сообщений;
- реализована регистрация и авторизация клиентов с помощь spring security;
- разработано клиентское приложение с использованием Material Design.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Николай Прохоренок, Владимир Дронов. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. — СПб.: «БХВ-Петербург», 2015. — С. 20-348.

2. Леон Шкляр, Рич Розен. Архитектура веб-приложений. — М.: «Эксмо», 2010. — С.35-150.

3. Эрик Фримен, Элизабет Робсон. Изучаем программирование на JavaScript. — СПб.: «Питер», 2018. — С.125-340.

4. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. — «Springer Science & Business Media», 2011. — С. 52-340.

5. Banks A., Porcello E. Learning React: functional web development with React and Redux. – " O'Reilly Media, Inc.", 2017.

6. Трепачёв Д.П. Учебник по фреймворку VUE //Аллея науки. – 2020. – Т. 1. – №. 7. – С. 928-930.

7. Дж. Уорсли, Дж. Дрейк. PostgreSQL . Для профессионалов — СПб.: «Питер», 2003. — С. 23-320.

8. Хэррон Д. Node. js Разработка серверных веб-приложений на JavaScript. – Litres, 2017.

9. Ефремова О. С. Требования охраны труда при работе на персональных электронно-вычислительных машинах. – М.: Альфа-Пресс, 2008.

ПРИЛОЖЕНИЯ

Приложение А Листинги программы на JavaScript

Файл «build.grable»

```
plugins {
     id 'org.springframework.boot' version '2.4.0-SNAPSHOT'
     id 'io.spring.dependency-management' version '1.0.9.RELEASE'
     id 'java'
}

group = 'org.kozhayev'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
     mavenCentral()
     maven { url 'https://repo.spring.io/milestone' }
     maven { url 'https://repo.spring.io/snapshot' }
}

dependencies {
     implementation 'org.springframework.boot:spring-boot-
starter-web'
     developmentOnly 'org.springframework.boot:spring-boot-
devtools'
     testImplementation('org.springframework.boot:spring-boot-
starter-test') {
          exclude group: 'org.junit.vintage', module: 'junit-
vintage-engine'
     }

     compile group: 'org.springframework.boot', name: 'spring-
boot-starter-thymeleaf', version: '2.3.0.RELEASE'

    compile group: 'javax.mail', name: 'javax.mail-api',
version: '1.6.2'
    compile group: 'com.sun.mail', name: 'javax.mail', version:
'1.6.2'

    compile group: 'org.springframework.boot', name: 'spring-
boot-starter-data-jpa', version: '2.3.0.RELEASE'
    compile group: 'org.postgresql', name: 'postgresql',
version: '42.2.12'
    compile group: 'javax.xml.bind', name: 'jaxb-api', version:
'2.3.1'
```

```
    compile group: 'org.glassfish.jaxb', name: 'jaxb-core',
version: '2.3.0.1'
    compile group: 'javax.xml', name: 'jaxb-impl', version:
'2.1'
    compile group: 'org.projectlombok', name: 'lombok', version:
'1.18.10'

     compile group: 'org.springframework.boot', name: 'spring-
boot-starter-security', version: '2.3.0.RELEASE'
}

test {
     useJUnitPlatform()
}
```

Файл «package.json»

```
{
  "name": "emailPGU",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "start": "webpack-dev-server"
  },
  "devDependencies": {
    "@babel/core": "^7.10.2",
    "@babel/preset-env": "^7.10.2",
    "babel-loader": "^8.1.0",
    "css-loader": "^3.5.3",
    "vue-loader": "^15.9.2",
    "vue-style-loader": "^4.1.2",
    "vue-template-compiler": "^2.6.11",
    "webpack": "^4.43.0",
    "webpack-cli": "^3.3.11",
    "webpack-dev-server": "^3.11.0"
  },
  "dependencies": {
    "vue": "^2.6.11",
    "vue-resource": "^1.5.1",
    "vue-router": "^3.3.2",
    "vuetify": "^2.2.32"
  }
}
```

Файл «webpack.config.js »

```
const path = require('path');
const VueLoaderPlugin = require('vue-loader/lib/plugin');
```

```
module.exports = {
    mode: 'development',
    devtool: 'source-map',
    entry: path.join(__dirname, 'src', 'main', 'resources',
'static', 'js', 'main.js'),
    devServer: {
        contentBase: './dist',
        compress: true,
        port: 8010,
        allowedHosts: [
            'localhost:8080'
        ],
        stats: 'errors-only',
        clientLogLevel: 'error'
    },
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /(node_modules|bower_components)/,
                use: {
                    loader: 'babel-loader',
                    options: {
                        presets: ['@babel/preset-env']
                    }
                }
            },
            {
                test: /\.vue$/,
                loader: 'vue-loader'
            },
            {
                test: /\.css$/,
                use: [
                    'vue-style-loader',
                    'css-loader'
                ]
            }
        ]
    },
    plugins: [
        new VueLoaderPlugin()
    ],
    resolve: {
        modules: [
            path.join(__dirname, 'src', 'main', 'resources',
'static', 'js'),
            path.join(__dirname, 'node_modules'),
        ],
    }
}
```

Файл «EmailPGUApplication.java »

```
package org.kozhayev.emailPGU;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmailPguApplication {

     public static void main(String[] args) {
          SpringApplication.run(EmailPguApplication.class, args);
     }

}
```

Файл «MainController.java»

```
package org.kozhayev.emailPGU.controller;

import org.kozhayev.emailPGU.domain.Email;
import org.kozhayev.emailPGU.domain.User;
import org.kozhayev.emailPGU.service.MyUserPrincipal;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.mail.*;
import javax.mail.internet.InternetAddress;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Properties;

@Controller
@RequestMapping("/")
public class MainController {

    @GetMapping
        public String main(Model model) {
        HashMap<Object, Object> data = new HashMap<>();
        try{
```

```
            MyUserPrincipal userPrincipal = (MyUserPrincipal)
SecurityContextHolder.getContext().getAuthentication().getPrinci
pal();

            data.put("profile", userPrincipal.getUsername());
            data.put("outbox",   findEmail("Sent",
userPrincipal.getUsername(), userPrincipal.getPassword()));
            data.put("inbox", findEmail("INBOX",
userPrincipal.getUsername(), userPrincipal.getPassword()));


        }catch (Exception e){
            data.put("profile", false);
        }

//        data.put("messages", messageRepo.findAll());

        model.addAttribute("frontendData", data);
//        model.addAttribute("isDevMode",
"dev".equals(profile));


        return "index";
    }

    private List<Email> parseMessage(Message[] messages){
        try {
            List<Email> inbox = new ArrayList<>();
            int size = 0;
            if (messages.length > -1 && messages.length-21 < 0)
               size = messages.length;
            else
                size = messages.length-21;
            int end = size < 20 ? 0 : size - 20;

            for (int i = size - 1 ; i >= end; i--) {
                Email email = new Email();

email.setToSend(((InternetAddress)messages[i].getFrom()
[0]).getAddress());

email.setFromSend(((InternetAddress)messages[i].getAllRecipients
()[0]).getAddress());
                email.setSubject(messages[i].getSubject());

email.setText( messages[i].getContent().toString());
                inbox.add(email);
            }
            return inbox;
        } catch (MessagingException e) {
            e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    private List<Email> findEmail(String nameFolder, String
email, String password) {
        Properties properties = new Properties();
        Folder folder = null;
        Store store = null;
        try{

properties.setProperty("mail.imap.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
            Session session =
Session.getDefaultInstance(properties);
            store = session.getStore("imap");
            store.connect("imap." + email.split("@")[1], 993,
email, password);
            //Folder[] f = store.getDefaultFolder().list();

            folder = store.getFolder(nameFolder);
            folder.open(Folder.READ_ONLY);
            int count = folder.getMessageCount();
            return parseMessage(folder.getMessages(1, count));

        } catch (Exception e) {
            e.printStackTrace();
        } finally {

            if (folder != null) {
                try {
                    folder.close(false);
                } catch (MessagingException e) {
                    e.printStackTrace();
                }
            }
            if (store != null) {
                try {
                    store.close();
                } catch (MessagingException e) {
                    e.printStackTrace();
                }
            }
        }
        return null;
    }

}
```

Файл «RegistrationConreoller.java»

```
package org.kozhayev.emailPGU.controller;


import org.kozhayev.emailPGU.domain.User;
import org.kozhayev.emailPGU.repo.UserDetailsRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import javax.mail.*;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Properties;

@RestController
@RequestMapping("/registration")
public class RegistrationController {

    private UserDetailsRepo userDetailsRepo;

    @Autowired
    public RegistrationController(UserDetailsRepo
userDetailsRepo) {
        this.userDetailsRepo = userDetailsRepo;
    }

    @PostMapping
    public int registration(Model model, @RequestBody User user)
{

        if (userDetailsRepo.findByEmail(user.getEmail()) ==
null){

            boolean find = findEmail(user.getEmail(),
user.getPassword());
            if (find) {
                user.setLastVisit(LocalDateTime.now());
                userDetailsRepo.save(user);
            }
            return find ? 1 : -1;

        } else {
            return 0;
        }
    }

    private boolean findEmail(String email, String password) {
        Properties properties = new Properties();
        Folder inbox = null;
```

```
        Store store = null;
        try{

properties.setProperty("mail.imap.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
            Session session =
Session.getDefaultInstance(properties);
            store = session.getStore("imap");
            store.connect("imap." + email.split("@")[1], 993,
email, password);
            inbox = store.getFolder("INBOX");
        } catch (Exception e) {
            return false;
            //e.printStackTrace();
        } finally {

//            if (inbox != null) {
//                try {
//                    inbox.close(false);
//                } catch (MessagingException e) {
//                    e.printStackTrace();
//                }
//            }
            if (store != null) {
                try {
                    store.close();
                } catch (MessagingException e) {
                    return false;
                    //e.printStackTrace();
                }
            }
        }
        return true;
    }
}
```

Файл «EmailController.java»

```
package org.kozhayev.emailPGU.controller;

import org.kozhayev.emailPGU.domain.Email;
import org.kozhayev.emailPGU.domain.User;
import org.kozhayev.emailPGU.repo.UserDetailsRepo;
import org.kozhayev.emailPGU.service.MyUserPrincipal;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

```
import java.util.Properties;
import javax.mail.*;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Properties;

@RestController
@RequestMapping("/send-email")
public class EmailController {

    @PostMapping
    public int sendEmail (@RequestBody Email email){
        Properties properties = new Properties();
        MyUserPrincipal userPrincipal = (MyUserPrincipal)
SecurityContextHolder.getContext().getAuthentication().getPrinci
pal();

        properties.put("mail.smtp.host", "smtp." +
email.getToSend().split("@")[1]);
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.socketFactory.port", "465");
        properties.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
        Session session = Session.getDefaultInstance(properties,
                //Аутентификатор - объект, который передает
логин и пароль
                new Authenticator() {
                    @Override
                    protected PasswordAuthentication
getPasswordAuthentication() {
                        return new
PasswordAuthentication(userPrincipal.getUsername(),
userPrincipal.getPassword());
                    }
                });
        Message message = new MimeMessage(session);
        //От кого
        try {
            message.setFrom(new
InternetAddress(userPrincipal.getUsername()));

            message.setRecipient(Message.RecipientType.TO, new
InternetAddress(email.getToSend()));

            //Тема письма
            message.setSubject(email.getSubject());
            //Текст письма
            message.setText(email.getText());
        } catch (MessagingException e) {

        }
```

```
            //Поехали!!!
        try {
            Transport.send(message);
        } catch (MessagingException e) {
            e.printStackTrace();
        }

        return 1;
    }
}
```

Файл «WebMvcConfig.java»

```
package org.kozhayev.emailPGU.config;

import
org.springframework.boot.web.server.ConfigurableWebServerFactory
;
import org.springframework.boot.web.server.ErrorPage;
import
org.springframework.boot.web.server.WebServerFactoryCustomizer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigur
er;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Bean
    public
WebServerFactoryCustomizer<ConfigurableWebServerFactory>
webServerCustomizer(){
        return container -> {
            container.addErrorPages(new
ErrorPage(HttpStatus.NOT_FOUND, "/"));
            container.addErrorPages(new
ErrorPage(HttpStatus.METHOD_NOT_ALLOWED, "/"));
        };
    }
}
```

Файл «WebSecurityConfig.java»

```
package org.kozhayev.emailPGU.config;

import com.sun.security.auth.PrincipalComparator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import
org.springframework.security.config.annotation.authentication.bu
ilders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration
.WebSecurityConfigurerAdapter;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UsernameNotFoundEx
ception;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import
org.springframework.security.crypto.password.NoOpPasswordEncoder
;
import
org.springframework.security.crypto.password.PasswordEncoder;
import javax.sql.DataSource;

import java.security.Principal;
import java.time.LocalDateTime;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {

    @SuppressWarnings("deprecation")
    @Bean
    public static NoOpPasswordEncoder passwordEncoder() {
        return (NoOpPasswordEncoder)
NoOpPasswordEncoder.getInstance();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception
{
        http
                .antMatcher("/**")
                .authorizeRequests()
                .antMatchers("/", "/login**", "/registration**",
"/welcome**", "/js/**", "/error**").permitAll()
                .anyRequest().authenticated()
```

```
                .and()
                .formLogin().permitAll()

.and().logout().logoutSuccessUrl("/").permitAll()
                .and()
                .csrf().disable();
    }
}
```

Файл «User.java»

```
package org.kozhayev.emailPGU.domain;

import lombok.Data;
import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "usr")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String email;
    private String password;
    private String name;
    private String userpic;
    private LocalDateTime lastVisit;


    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getUserpic() {
        return userpic;
    }
```

```
    public void setUserpic(String userpic) {
        this.userpic = userpic;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public LocalDateTime getLastVisit() {
        return lastVisit;
    }

    public void setLastVisit(LocalDateTime lastVisit) {
        this.lastVisit = lastVisit;
    }
}
```

Файл «Email.java»

```
package org.kozhayev.emailPGU.domain;

public class Email {

    private String toSend;
    private String fromSend;
    private String subject;
    private String text;

    public String getFromSend() {
        return fromSend;
    }

    public void setFromSend(String fromSend) {
        this.fromSend = fromSend;
    }

    public String getToSend() {
        return toSend;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
```

```
    public void setToSend(String toSend) {
        this.toSend = toSend;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

Файл «UserDetailsRepo.java»

```
package org.kozhayev.emailPGU.repo;

import org.kozhayev.emailPGU.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserDetailsRepo extends JpaRepository<User,
Long> {
    User findByEmail(String email);
}
```

Файл «MyUserDetailsService.java»

```
package org.kozhayev.emailPGU.service;

import org.kozhayev.emailPGU.domain.User;
import org.kozhayev.emailPGU.repo.UserDetailsRepo;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.core.userdetails.UsernameNotFoundEx
ception;
import org.springframework.stereotype.Service;



@Service
public class MyUserDetailsService implements UserDetailsService
{

    @Autowired
```

```
    private UserDetailsRepo userRepo;

    @Override
    public UserDetails loadUserByUsername(String username) {
        User user = userRepo.findByEmail(username);
        if (user == null) {
            throw new UsernameNotFoundException(username);
        }
        return new MyUserPrincipal(user);
    }
}
```

Файл «MyUserPrincipal.java»

```
package org.kozhayev.emailPGU.service;

import org.kozhayev.emailPGU.domain.User;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;

public class MyUserPrincipal implements UserDetails {

    private User user;

    MyUserPrincipal(User user) {
        this.user = user;
    }

    public User getUser() {
        return user;
    }

    @Override
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return null;
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getEmail();
```

```
    }

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

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

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

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

Файл «application.properties»

```
spring.datasource.url=jdbc:postgresql://localhost:5000/kozhayevE
Mail
spring.datasource.username=postgres
spring.datasource.password=123
spring.jpa.generate-ddl=true

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation
=true
```

Приложение В Листинг FrontEnd

Файл «index.html»

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <link href="https://fonts.googleapis.com/css?
family=Roboto:100,300,400,500,700,900" rel="stylesheet">
    <meta name="viewport" content="width=device-width, initial-
scale=1, maximum-scale=1, user-scalable=no, minimal-ui">
    <link
href="https://cdn.jsdelivr.net/npm/@mdi/font@5.x/css/materialdes
ignicons.min.css" rel="stylesheet">
    <link
href="https://cdn.jsdelivr.net/npm/vuetify@2.x/dist/vuetify.min.
css" rel="stylesheet">

<title>E-mail "Слава"</title>
<script th:inline="javascript">
    var frontendData = [[${frontendData}]];
</script>
</head>

<body>
    <div id="app"></div>
    <script src="http://localhost:8010/main.js"></script>
</body>
</html>
```

Файл «main.js»

```
import Vue from 'vue'
import VueRouter from "vue-router"
import VueResource from 'vue-resource'
import App from './pages/App.vue'
import Vuetify from 'vuetify'
import 'vuetify/dist/vuetify.min.css'

Vue.use(Vuetify)
Vue.use(VueResource)
Vue.use(VueRouter)

new Vue({
    el: '#app',
    vuetify: new Vuetify(),
    router: new VueRouter({
        mode: 'history',
```

```
    }),
    render: a => a(App)
})
```

Файл «App.vue»

```
<template>
    <v-app>
        <Welcome v-if="!profile && this.$route.path ===
'/welcome'"></Welcome>
        <Menu v-if="this.$route.path === '/' && profile"></Menu>
        <div v-else-if="this.$route.path === '/' ">
            <meta http-equiv="refresh" content="1;
url=http://localhost:8080/welcome">
        </div>
        <Registration v-if="this.$route.path ===
'/registration'"></Registration>
    </v-app>
</template>

<script>
    import Registration from "./Registration.vue";
    import Menu from "./Menu.vue";
    import Welcome from "./Welcome.vue";

    export default {
        name: "App",
        components: {
            Registration,
            Menu,
            Welcome,
        },
        data:() =>({
            profile: frontendData.profile,
        }),
        methods: {
            welcome(){
                document.location.href = "/welcome";
                return true
            }
        }
    }

</script>

<style scoped>

</style>
```

Файл «Menu.vue»

```
<template>
    <div>
        <v-app-bar app clipped-left>
            <v-app-bar-nav-icon @click.stop="drawer = !
drawer"></v-app-bar-nav-icon>
            <v-toolbar-title>Почта Славы</v-toolbar-title>
            <v-spacer></v-spacer>
            <div class="mx-5">{{profile}}</div>
            <a href="/logout">  Выйти</a>
        </v-app-bar>

        <v-navigation-drawer
                v-model="drawer"
                app
                clipped
        >
            <v-list dense>

                <v-list-item link v-on:click="currentTab='send-
email'">
                    <v-list-item-action  >
                        <v-icon>send</v-icon>
                    </v-list-item-action>
                    <v-list-item-content>
                        <v-list-item-title>Написать
сообщениее</v-list-item-title>
                    </v-list-item-content>
                </v-list-item>

                <v-list-item link v-on:click="currentTab='input-
email'">
                    <v-list-item-action  >
                        <v-icon>email</v-icon>
                    </v-list-item-action>
                    <v-list-item-content>
                        <v-list-item-title>Входящие</v-list-
item-title>
                    </v-list-item-content>
                </v-list-item>

                <v-list-item link v-
on:click="currentTab='output-email'">
                    <v-list-item-action>
                        <v-icon>email</v-icon>
                    </v-list-item-action>
                    <v-list-item-content>
                        <v-list-item-title>Отправленные</v-list-
item-title>
                    </v-list-item-content>
                </v-list-item>
            </v-list>
```

```
        </v-navigation-drawer>

        <<v-content class="ma-10">
            <component v-
bind:is="currentTabComponent"></component>
        </v-content>

        <v-footer app>
            <span>&copy; 2020</span>
        </v-footer>
    </div>
</template>

<script>
    import SendEmail from "components/SendEMail.vue"
    import OutputEmail from "components/OutputEMail.vue"
    import InputEmail from "components/InputEmail.vue"

    export default {
        name: "Menu",
        components:{
            SendEmail,
            OutputEmail,
            InputEmail,
        },
        data: () => ({
            currentTab: "send-email",
            tabs: ["input-email", "output-email", "send-email"],
            drawer: null,
            profile: frontendData.profile,
        }),
        computed: {
            currentTabComponent: function () {
                return this.currentTab.toLowerCase();
            }
        }
    }
</script>

<style scoped>

</style>
```

Файл «Welcome.vue»

```
<template>
    <div>
        <v-app-bar app clipped-left>
            <v-toolbar-title>Почта Славы</v-toolbar-title>
        </v-app-bar>
        <v-content>
            <v-layout  column align-center justify-center>
                <h3 class="mt-12">Добро пожаловать, дорогой
друг!</h3>
                <v-layout>
                    <v-btn class="ma-10" tile x-large
color="yellow" href="/registration">Зарегистрироваться</v-btn>
                    <v-btn class="ma-10" tile x-large
color="white" href="/login">Войти</v-btn>
                </v-layout>
            </v-layout>
        </v-content>
        <v-footer app>
            <span>&copy; 2020</span>
        </v-footer>
    </div>
</template>

<script>
    export default {
        name: "Welcome"
    }
</script>

<style scoped>

</style>
```

Файл «Registration.vue»

```
<template>
    <v-content>
        <v-container class="fill-height" fluid >
            <v-row align="center" justify="center">
                <v-col cols="12" sm="8" md="4">
                    <v-card class="elevation-12 mb-7" >
                        <v-toolbar color="primary" dark flat>
                            <v-toolbar-title>Форма
Регистрации</v-toolbar-title>
                            <v-spacer></v-spacer>
                        </v-toolbar>
                        <v-card-text>
                            <v-form>
```

```
                                    <v-text-field
                                            id="name"
                                            label="name"
                                            name="name"
                                            prepend-icon="mdi-
account"
                                            type="text"
                                            v-model="name"
                                    ></v-text-field>
                                    <v-text-field
                                            id="email"
                                            label="email"
                                            name="email"
                                            prepend-icon="mdi-
account"
                                            type="text"
                                            v-model="email"
                                    ></v-text-field>
                                    <v-text-field
                                            id="password"
                                            label="Password"
                                            name="password"
                                            prepend-icon="mdi-lock"
                                            type="password"
                                            v-model="password"
                                    ></v-text-field>
                                </v-form>
                            </v-card-text>
                            <v-card-actions>
                                <v-spacer></v-spacer>
                                <v-btn color="primary"
:loading=loading :disabled=disabled
@click="regProfile">Зарегистрировать</v-btn>
                            </v-card-actions>
                        </v-card>
                        Уже зарегистрированы?
                        <v-btn text color="primary"
@click="login">Войти</v-btn>
                    </v-col>
                </v-row>
            </v-container>
        </v-content>
    </template>

    <script>
        export default {
            name: "Menu",
            props: {
                source: String,
            },
            data(){
```

```
            return{
                name: '',
                email: '',
                password: '',
                loading: false,
                disabled: false,
            }

        },
        methods: {
            regProfile() {
                this.loading = true;
                this.disabled = true;
                const message = {email: this.email, password:
this.password, name: this.name}
                this.$resource('/registration').save({},
message).then(result =>
                    result.json().then(data => {
                        switch (data) {
                            case (1):
                                alert("Регистрация прошла
успешна, авторизуйтесь")
                                document.location.href =
"/login"
                                break;
                            case (0):
                                alert("Пользователь уже
зарегистрирован, авторизуйтесь")
                                document.location.href =
"/login"
                                break;
                            case (-1):
                                alert("Данные не прошли
проверку, повторите попытку регистрации")
                                this.email = ''
                                this.password = ''
                                break;
                            }
                        this.loading = false;
                        this.disabled = false;

                        }
                    ))

            },
            login(){
                document.location.href = "/login"
            }
        }
    }
</script>
```

```
<style scoped>

</style>
```

Файл «InputEmail.vue»

```
<template>
    <div>
        Ввходящие сообщения
        <v-layout column>
            <OneLetter v-for="letter in letters"
:email="letter.toSend" :subject="letter.subject" />
        </v-layout>
    </div>
</template>

<script>
    import OneLetter from "components/OneLetter.vue"

    export default {
        name: "InputEmail",
        components:{
            OneLetter,
        },
        data: () => ({
            letter: null,
            letters: frontendData.inbox,
            drawer: null,
        })
    }
</script>

<style scoped>

</style>
```

Файл «OutputEmail.vue»

```
<template>
    <div>
        Отправленные сообщения
        <v-layout column>
            <OneLetter v-for="letter in letters"
:email="letter.fromSend" :subject="letter.subject" />
        </v-layout>
    </div>
</template>

<script>
```

```
    import OneLetter from "components/OneLetter.vue"

    export default {

        name: "OutputEmail",
        // props: {
        //      mail: frontendData
        // },
        components:{
            OneLetter,
        },

        data: () => ({
            letter: null,
            letters: frontendData.outbox,
            drawer: null,
        })
    }
</script>

<style>

</style>
```

Файл «OneLetter.vue»

```
<template>
    <div class="my-2">
        Email: <h4>{{ email }}</h4> Тема: {{ subject }}
    </div>
</template>

<script>
    export default {
        name: "OneLetter",
        props: ['subject',  'email']
    }
</script>

<style scoped>

</style>
```

Файл «SendEmail.vue»

```
<template>
    <div>
        <v-text-field
                name="toSend"
                v-model="toSend"
                class="pa-2 ma-2"
```

```
                label="Кому отправить"
                :rules="rulesEMail"
                hide-details="auto"
        ></v-text-field>
        <v-text-field
                name="subject"
                v-model="subject"
                class="pa-2 ma-2"
                label="Тема сообщения"
                :rules="rulesText"
                hide-details="auto"
        ></v-text-field>
        <v-textarea
                v-model="text"
                class="my-4"
                :rules="rulesText"
                :auto-grow=false
                :clearable=true
                :counter=0
                :filled=true
                :flat=false
                :label="'Сообщение'"
                :loading=false
                :no-resize=false
                :outlined=true
                :persistent-hint=true
                :placeholder="'Введите текст сообщения'"
                :rounded=false
                :row-height=24
                :rows=10
                :shaped=true
                :single-line=false
                :solo=false
        ></v-textarea>
        <v-btn class="ma-2" :disabled=disabled :loading=loading
outlined color="indigo" @click="sendMail">Отправить</v-btn>
    </div>
</template>

<script>
    export default {
        name: "SendEMail",
        data: () =>({
            rulesEMail:[
                value => !!value || 'Required.',
                value => (value && value.length >= 3) || 'Min 3
characters',
                value => (value && value.toString().match(/.+@.
+\..+/i) != null) || 'This not e-Mail.',
            ],
            rulesText:[
```

```
                value => !!value || 'Required.',
            ],
            toSend: '',
            text: '',
            subject: '',
            loading: false,
            disabled: false,
        }),
        methods: {
            sendMail() {
                const message = {toSend: this.toSend, subject:
this.subject, text: this.text}

                if (this.toSend === "" || this.text === ""){
                    alert("Заполните обязательные поля
правильно")
                } else {
                    this.loading = true
                    this.disabled = true
                    this.$resource('/send-email').save({},
message).then(result =>
                        result.json().then(data => {
                                switch (data) {
                                    case (1):
                                        alert("Сообщение успешно
отправлено")
                                        break;
                                    case (-1):
                                        alert("Сообщение не
отправлено")
                                        break;
                                }
                            this.loading = false
                            this.disabled = false
                            }
                        ))
                }
            },
        }
    }
</script>
<style scoped>
</style>
```

Приложение D Презентация

Система приема и отправки почтовых сообщений

Автор работы: студент группы 16ВВ2 Кожаев В.Д.

Научный руководитель к.т.н., доцент Дубравин А.В.

Цель и объекты работы

Цель работы: разработать систему приема и отправки почтовых сообщений.

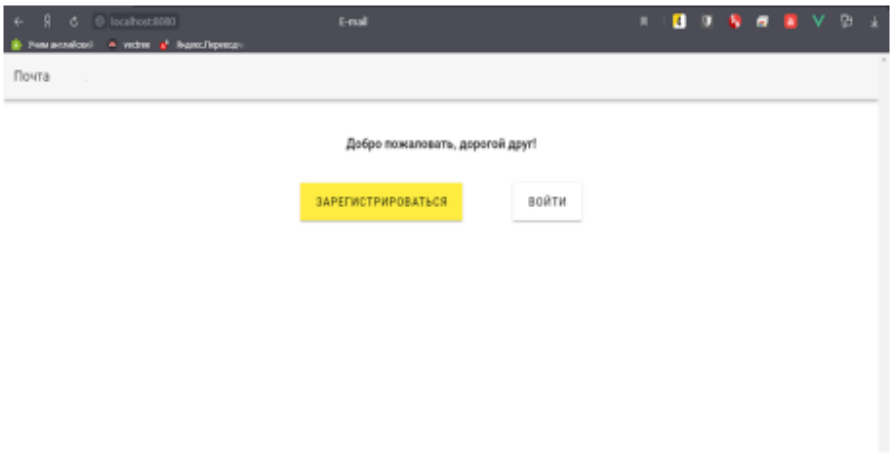
Объект работы: клиент серверное приложение выполняющие следующие функции:

- Прием почтовых сообщений
- Хранение сообщений
- Хранение данных о пользователях
- Отправка сообщений

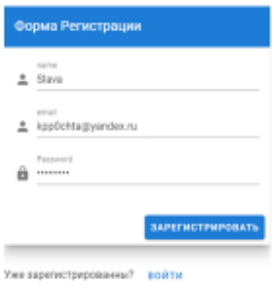
Стек используемых технологий

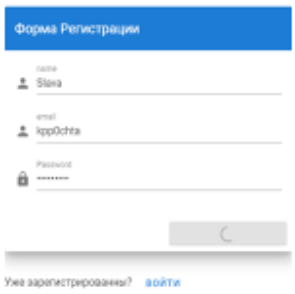
UML диаграмма классов

Результат работы страницы «Приветствие»

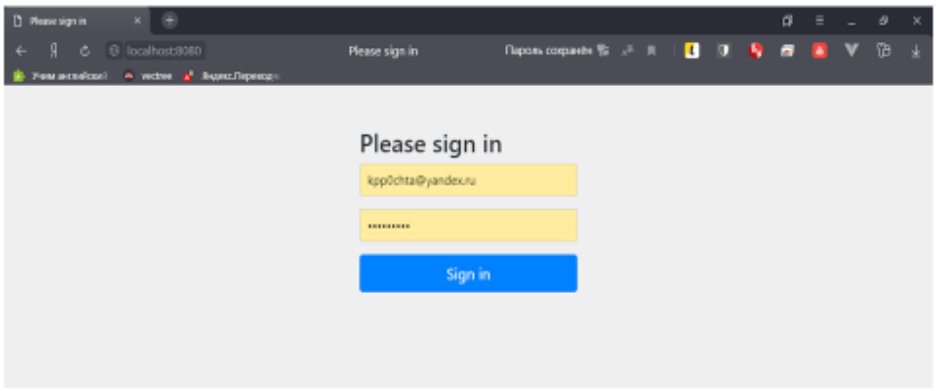


Результат работы страницы «Регистрация»

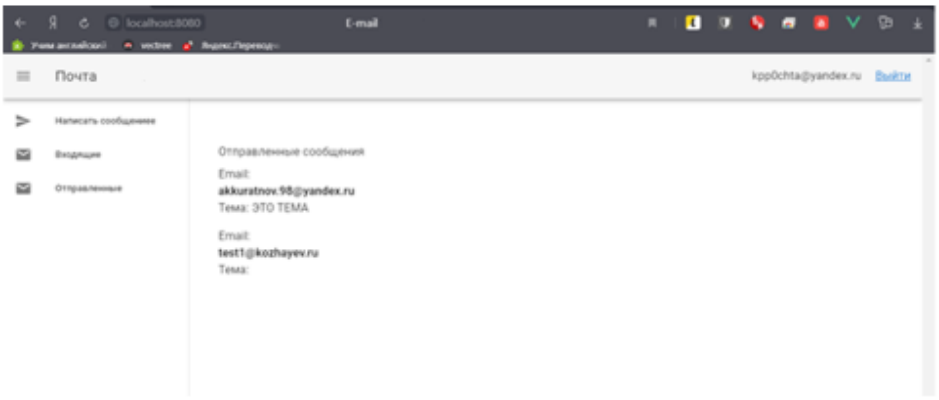




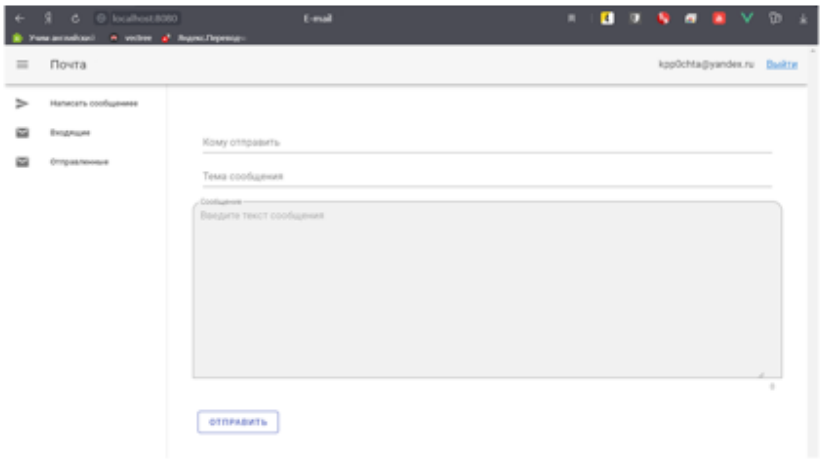
Результат работы страницы «Авторизации»



Результат работы страницы «Отправленных сообщений»



Результат работы страницы «Новое сообщение»



Общий результат проведенной работы

1. Проведен анализ современных способов разработки приложений.
2. Проведен анализ популярных технологических и архитектурных решений для отображения представления и оптимизации современных приложений
3. Разработана и отлажена система приема и отправки почтовых сообщений

Спасибо за внимание