Содержание

[Введение 3](#_Toc119235275)

[1. Анализ объекта 4](#_Toc119235276)

[1.1 Описание предметной области. 4](#_Toc119235277)

[1.2 Построение концептуальной модели предметной области. 6](#_Toc119235278)

[2 Постановка задачи 7](#_Toc119235279)

[2.1 Определение требований к программной системе. 7](#_Toc119235280)

[2.2 Описание аналогов системы. 8](#_Toc119235281)

[2.3 Обзор и обоснование выбора средств реализации 9](#_Toc119235282)

[3 Проектирование 10](#_Toc119235283)

[3.1 Разработка архитектуры программного продукта 10](#_Toc119235284)

[3.2 Проектирование структур хранения данных. 17](#_Toc119235285)

[4 Реализация 20](#_Toc119235286)

[4.1 Разработка архитектуры программного продукта 20](#_Toc119235287)

[4.2 Разработка интерфейса программного продукта 28](#_Toc119235288)

[5. Тестирование 36](#_Toc119235289)

[Литература 40](#_Toc119235290)

[Приложение А 42](#_Toc119235291)

# Введение

Практически все пользователи сети Интернет обмениваются сообщениями в том или ином виде. Широко используется электронная почта – для переписки и рассылки данных: документов, фото или видео. Не менее востребованы чаты – с этим способом связи сталкивается каждый при посещении страниц ежедневно.

Чат — средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение. Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает чат от форумов и других «медленных» средств.

Графический способ решения задач линейного программирования целесообразно использовать для решения задач с двумя переменными, когда ограничения выражены неравенствами.

Мой проект представляет собой чат программу, которая имеет возможности регистрации пользователей и общения в чатах. Данная курсовая работа выполнена на языке программирования высокого уровня Java с использованием IDE IntelijIDEA.

# Анализ объекта

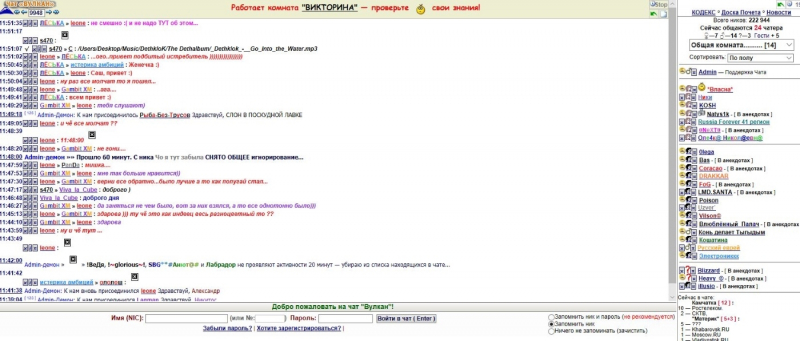
## **Описание предметной области.**

Чат как метод обмена мгновенными сообщениями появился в конце 80-х годов XX века. Сначала был разработан протокол IRC (аббревиатура переводится так: ретранслируемый интернет-разговор). Он позволял отправлять сообщения как группе пользователей, так и одному человеку. Следом появляются системы обмена мгновенными сообщениями – мессенджеры. Чат-культура способствовала активному применению специальных значков (смайлов) для придания тексту дополнительной эмоциональной окраски.

Во время общения применяются специальные смайлы для придания тексту дополнительной эмоциональной окраски

Пик популярности чатов пришелся на 90-е годы, общение с людьми, находящимися за много тысяч километров, привлекало пользователей – в отличие от телефонной связи чаты позволяли создавать целые сообщества по интересам или просто «интернет-болталки», в которых велись беседы на любые темы. Когда еще не было социальных сетей пользователи из России проводили много часов в интернет-клубах, общаясь в чатах: знакомились, обсуждали различные темы, искали работу и так далее.

До сих пор живы русскоязычные чаты: «Вулкан» – действует до сих пор. Есть несколько виртуальных комнат: общая, анекдоты, викторина, доска объявлений;



Рикусонк 1.1 - Интерфейс чата «Вулкан»

Важно заметить, что чат не стоит путать с форумом, поскольку для форумов не характерна привязка к общению в реальном времени. Сегодня пользуются популярностью не только текстовые виды чатов, но и – видеочаты. С помощью видеочатов пользователи могут видеть и разговаривать друг с другом в онлайн-режиме.

Для этого компьютер или смартфон участников должен быть оборудован камерой и микрофоном. Кроме этого, определенной востребованностью пользуются голосовые чаты. Например, во время компьютерных игр участники могут обмениваться голосовыми сообщениями в режиме «онлайн».

В настоящее время чаты имеют огромную популярность в соцсетях и мессенджерах. Пользователи могут организовывать приватные группы для общения в «WhatsApp», «Viber», «Telegram» и прочих мессенджерах. Также чаты есть и в таких программах для видеосвязи, как «Skype» и «Zoom».

## **Построение концептуальной модели предметной области.**

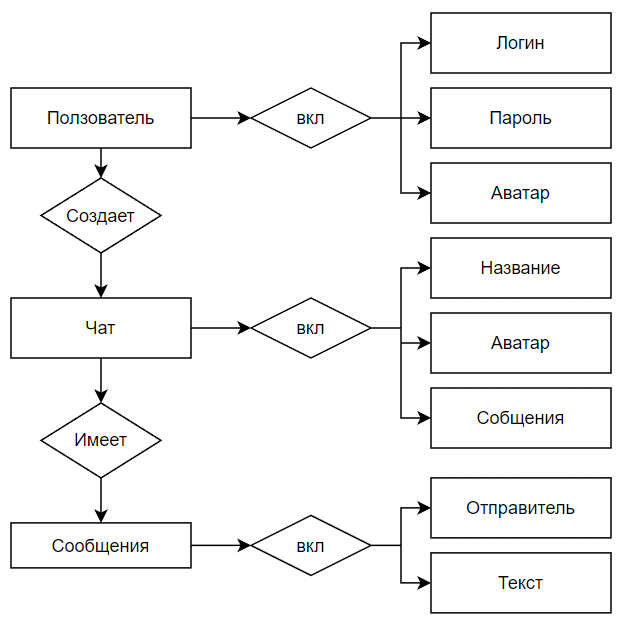


Рисунок 1.2 – Модель предметной области

Крнцептуальная модель чата состоит из рех сущностей – пользователя, чата и сообщений. Пользователь – это человек который общается. Для входа в систему он использует логин и пароль. Также имеет аватарку. Чат представляет собой место, где общаюся пользователи. Он характеризуется названием и аватаркой. Внутри час содержит сообщения – текст, отправленный пользователем..

# Постановка задачи

## **Определение требований к программной системе.**

Программная система должна удовлетворять нижеперечисленным требованиям и решать поставленные задачию Приложение должно:

1. иметь функцию регистрации пользователя
2. иметь функцию входа в систему
3. иметь функцию редактирования профиля
4. поддерживать ввод данных с помозью клавиатуры и мыши
5. иметь функциональность создавать и удалять чаты
6. иметь возможность пользователю писать сообщения в чате и получать новые сообщения в режиме реального времени
7. иметь дружелюбный интерфейс
8. использовать язык программирвоания Java версии 11

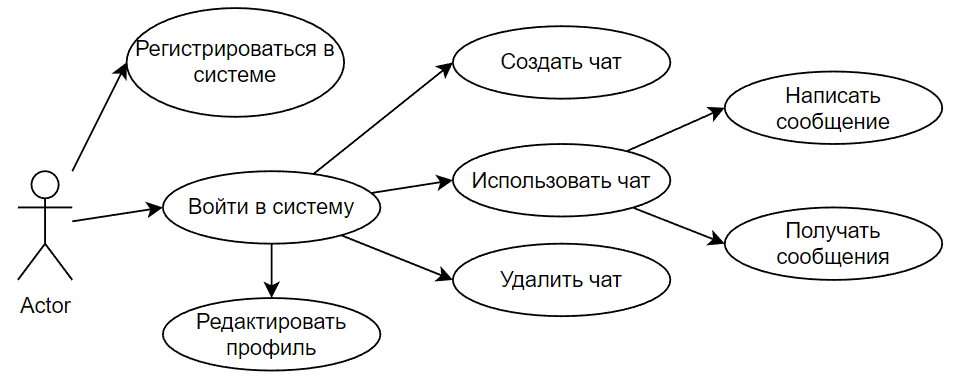


Рисунок 2.1 - Диаграмма вариантов использования

## **Описание аналогов системы.**

В сети Интернет можно найти большое количество работ, которые реализуют решение данных задач. Однако при поиске в 90% случаев в результат вы получите рекомендации онайн чата для сайта, который представляют собой консультацию по товарам, а не обычный чат для общения.

Мой курсовой проект представляет собой веб приложение, которое можно легко развернуть в системе и затем использовать всеми пользователями сети без обязательного доступа к интернету.

### **2.3 Обзор и обоснование выбора средств реализации**

При разработке программного средства будут использованы следующие технологии: язык программирования Java 11, сервер базы данных Postgres. В качестве фреймворка будет выбран фреймворк Spring Framework.

Для реализации Back-end части будут использована технология Spring Framework. Для реализации Front-end части будут использованы следующие технологии: Thymeleaf, HTML, CSS и JavaScript. Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Существует огромное количество инструментария для программирования как на стороне сервера, так и на стороне клиента, как платных, так и бесплатных.

Основные программные средства, которые будут использованы при разработке программного средства:

интегрированная среда разработки IDEA для разработки на языках программирования Java и JavaScript, языке разметки HTML и каскадных таблиц стилей CSS с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для Java;

браузеры Opera и Google Chrome для проверки на кроссбраузерность программного средства, а также проверки результатов работы программного средства при разработке.

Все перечисленные выше серверные и клиентские технологии, а также соответствующий инструментарий идеально подходят для разработки веб-приложений.

# Проектирование

## **Разработка архитектуры программного продукта**

Чаще всего веб-приложения состоят как минимум из трёх основных компонентов:

Клиентская часть веб приложения — это графический интерфейс. Это то, что вы видите на странице. Графический интерфейс отображается в браузере. Пользователь взаимодействует с веб-приложением именно через браузер, кликая по ссылкам и кнопкам.

Серверная часть веб-приложения — это программа или скрипт на сервере, обрабатывающая запросы пользователя (точнее, запросы браузера). Чаще всего серверная часть веб-приложения программируется на PHP. При каждом переходе пользователя по ссылке браузер отправляет запрос к серверу. Сервер обрабатывает этот запрос, вызывая некоторый PHP-скрипт, который формирует веб-страничку, описанную языком HTML, и отсылает клиенту по сети. Браузер тут же отображает полученный результат в виде очередной веб-страницы.

База данных (БД, или система управления баазми данных, СУБД) - программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. В случае форума или блога, хранимые в БД данные — это посты, комментарии, новости, и так далее. База данных располагается на сервере. Серверная часть веб-приложения (то есть, PHP скрипт) обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

Наше приложение также будет состоять из этих трех частей и основываться будет на Spring MVC. MVC — это не шаблон проекта, это конструкционный шаблон, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов [14]. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:

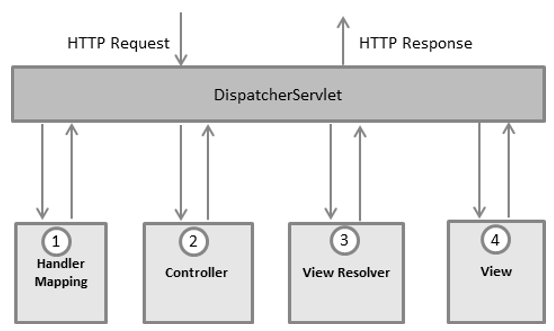


Рисунок 2.1 – Рабочий процесс обработки запроса

Ниже приведена последовательность событий, соответствующая входящему HTTP-запросу:

* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике, и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

DispatcherServlet отправляет запрос контроллерам для выполнения определённых функций. Аннотация @Controllerannotation указывает, что конкретный класс является контроллером. Аннотация @RequestMapping используется для мапинга (связывания) с URL для всего класса или для конкретного метода обработчика.

Аннотация Controller определяет класс как Контроллер Spring MVC. В первом случае, @RequestMapping указывает, что все методы в данном Контроллере относятся к URL-адресу "/hello".

Следующая аннотация @RequestMapping(method = RequestMethod.GET) используется для объявления метода printHello() как дефолтного метода для обработки HTTP-запросов GET. Вы можете определить любой другой метод как обработчик всех POST-запросов по данному URL-адресу.

Вы можете написать вышеуказанный Контроллер по-другому, указав дополнительные атрибуты для аннотации @RequestMapping следующим образом

@RequestMapping(value = "/hello", method = RequestMethod.GET)  
Атрибут «value» указывает URL, с которым мы связываем данный метод (value = "/hello"), далее указывается, что этот метод будет обрабатывать GET-запросы (method = RequestMethod.GET). Также, нужно отметить важные моменты в отношении приведённого выше контроллера:

* Вы определяете бизнес-логику внутри связанного таким образом служебного метода. Из него Вы можете вызывать любые другие методы.
* Основываясь на заданной бизнес-логике, в рамках этого метода Вы создаёте Модель (Model). Вы можете добавлять аттрибуты Модели, которые будут добавлены в Вид (View). В примере выше мы создаём Модель с атрибутом «message».
* Данный служебный метод возвращает имя Вида в виде строки String. В данном случае, запрашиваемый Вид имеет имя «hello».

Spring MVC поддерживает множество типов Видов для различных технологий отображения страницы. В том числе — JSP, HTML, PDF, Excel, XML, Velocity templates, XSLT, JSON, каналы Atom и RSS, JasperReports и проч. Но чаще всего используются шаблоны JSP, написанные при помощи JSTL или HTML файлы, используя Thymeleaf.

На рисунке 3.1 представлена диаграмма последовательности, которая представляет собой один конкретный экземпляр работы программы под управлением пользователя.

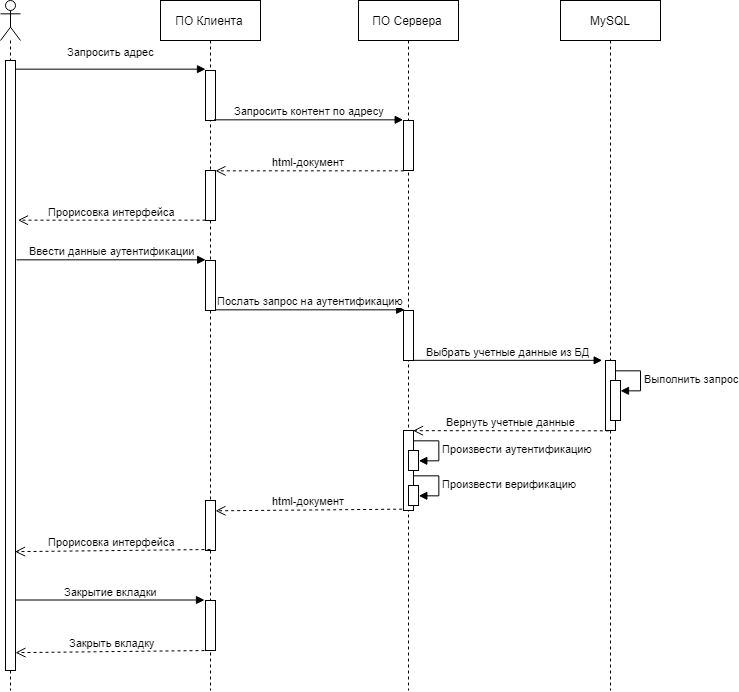


Рисунок 3.1 – Диаграмма последовательности

Диаграмма последовательности – это диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание – деятельность – уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

На диаграмме последовательности изображен один актёр – пользователь системы, а также следующие объекты:

– ПО Клиента;

– ПО Сервера;

– MySQL.

Представлена последовательность действия в три этапа:

1. Пользователь вводит адрес сайта в браузере, а сервер отправляет клиенту html-документ.

2. Клиент вводит данные аутентификации. Браузер посылает запрос серверу на аутентификацию, а сервер посылает запрос на сервер MySQL, который выбирает данные и посылает их серверу. Сервер их преобразует и возвращает клиентскому браузеру. Браузер отображает информацию для клиента.

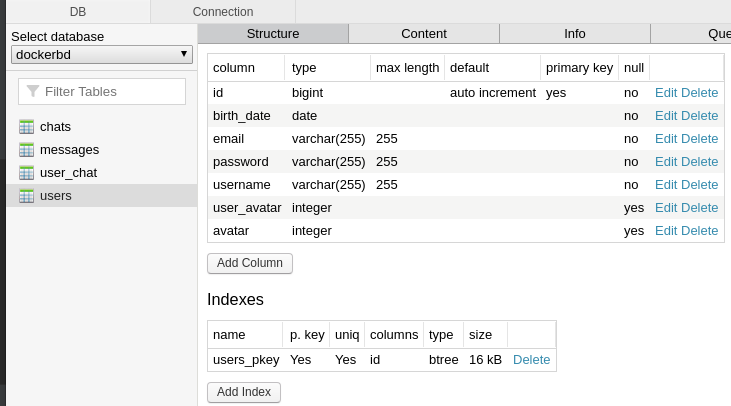
3. Клиент нажимает кнопку закрытия в браузере – вкладка закрывается. Работа системы завершена.

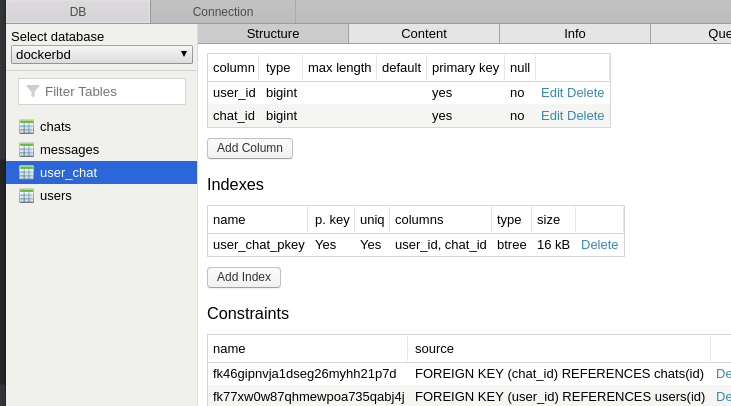
В этом подразделе произведено обоснование выбора технологий, а также описаны три основных типа диаграммы UML для отображения структуры данных

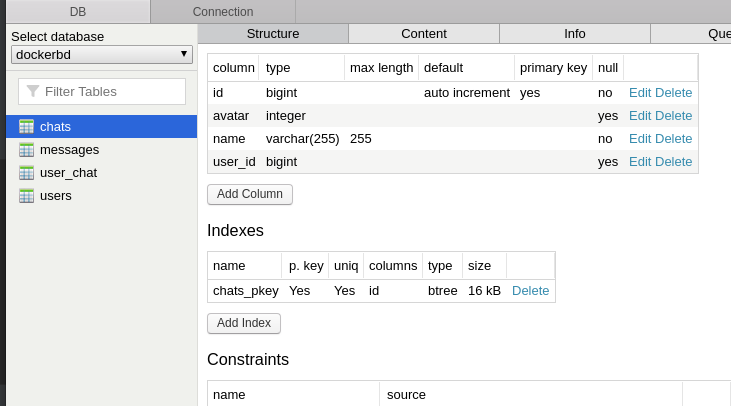
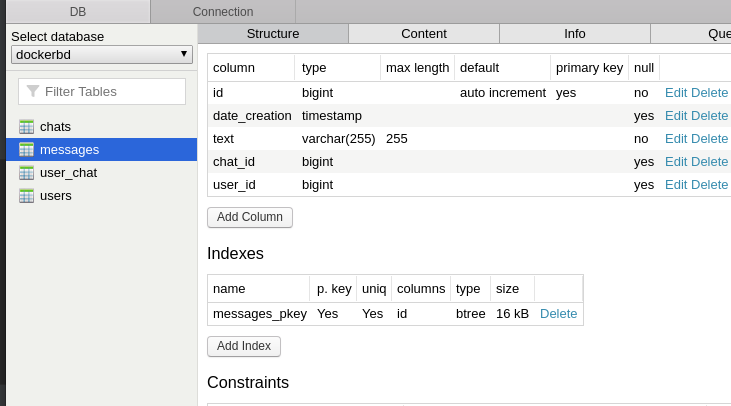
## **Проектирование структур хранения данных.**

Spring Framework позволяет генерировать базу данных на основе правильно описанных сущностей и взаимодействий между ними. Иначе говоря, после создания классов приложения, фреймворк при запуске приложения сам создаст базу данных, если она еще не была создана. Таблицами такой базы будут классы сущности, названиями таблиц будут названия сущностей. Тип и размер полей фреймворк также сам задаст на основе типов класса, если они не помечены аннотацией Length (задание размера вручную).

Схема базы данных отображена на рисунках ниже:







# Реализация

## **Разработка архитектуры программного продукта**

Так как мы используем шаблон MVC, идет отдельно разработка классов модели (сущности), отдельно вида (страниц для браузера) и контроллеров (обработки действий пользователя).

Так как данные в шаблоне будут изменяться по полученным из бд, нам нужно создать класс-сущность для хранения этих данных.

Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. Поля сущности совпадают с полями таблицы, в которой эти данные хранятся в базе данных.

Пример класса сущности отображен в листинге ниже.

Листинг 1. Сущность пользователя

@Entity

@Data

@Table(name = "Users")

public class User extends AbstractEntity {

@Column(nullable = false)

private String username;

@Column(nullable = false)

private String password;

@Column(nullable = false)

private String email;

@Column(nullable = false)

@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)

private LocalDate birthDate;

private UserAvatar avatar = UserAvatar.A1;

@JsonIgnore

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(name = "user\_chat",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "chat\_id"))

private Set<Chat> chats = new HashSet<>();

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

User user = (User) o;

return Objects.equals(username, user.username)

&& Objects.equals(password, user.password)

&& Objects.equals(email, user.email)

&& Objects.equals(birthDate, user.birthDate)

&& avatar == user.avatar;

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), username, password, email, birthDate, avatar);

}

@Override

public String toString() {

return "User{" +

"username='" + username + '\'' +

", password='" + password + '\'' +

", email='" + email + '\'' +

", birthDate=" + birthDate +

", avatar=" + avatar +

'}';

}

По подобию кода выше пишутся сущности для всех остальных объектов.

После написания сущностей идет описание сервисов – классов, которые взаимодействуют с базой данных, читая из нее данные и отдавая нам, или наоборот – записывая в нее.

Код классов для сервисов также похож между собой, так что покажем только абстрактную реализацию сервиса.

Листинг 2. Абстрактная реализация сервиса.

@RequiredArgsConstructor

public abstract class AbstractService<E> {

private final JpaRepository<E, Long> repository;

public E create(E entity) {

return repository.saveAndFlush(entity);

}

public List<E> read() {

return repository.findAll();

}

public Optional<E> findById(int id) {

return repository.findById((long) id);

}

@Transactional

public E update(E entity) {

return repository.saveAndFlush(entity);

}

public void delete(E entity) {

repository.delete(entity);

}

public void delete(int id) {

repository.deleteById((long) id);

}

}

После реализации сервисов, остается только 2 шага до окончания разработки приложения – это написание контроллеров и представлений. Контроллер – это специализированный класс, который отвечает на запросы пользователей по определенным ссылкам. Обычно они именуются так: СтраницаController, где страница – это имя страница, за которую отвечает контроллер. В контроллерах опишем ответы на запросы пользователя по взаимодействию с базой данных, т.е. ответы на запросы типа /read, /create/ edit. Ниже показан контроллер пользователя в качестве примера. Остальные контроллеры пишутся по его подобию.

Листинг 3. Контроллер входа и регистрации

@Controller

@RequiredArgsConstructor

public class LoginAndRegistrationController {

private final UserService userService;

@GetMapping("/login")

public String getLoginPage() {

return "login.html";

}

@PostMapping("/registration")

public String registerUser(@ModelAttribute User user,

Model model) {

Optional<User> byUsername = userService.findByUsername(user.getUsername());

if (byUsername.isPresent()) {

model.addAttribute("message", "User with same username already exists");

return "/login.html";

} else {

userService.create(user);

}

return "redirect:/";

}

}

Остался последний шаг – связывание шаблонов и контроллеров. Для этого воспользуемся шаблонизатором Thymeleaf, который работает в связке с Spring.

Суть его проста – те данные, что мы передавали в модель отображения, таймлиф можем выводить простой командой th:text=”${}”. В скобках пишется имя переменной. Обращение идет как к переменным класса. В нужных местах вставляем этот код. На нашем лендинге.

Так, для отображения шапки других шаблонов будут использоваться строки листинга 4.

Листинг 4. Форма профиля

<div class="container profile">

<div class="profileImage">

<img th:src="'/img/avatars/' + ${entity.avatar} + '.jpg'">

<div th:text="${entity.username} + '\'s profile'"></div>

</div>

<form class="form-signin" method="post" action="/profile">

<p>

<fieldset>

<legend>Select avatar:</legend>

<div>

<input type="radio" id="A1" name="avatars" value="1" required>

<label for="A1"><img src="/img/avatars/1.jpg"></label>

</div>

<div>

<input type="radio" id="A2" name="avatars" value="2">

<label for="A2"><img src="/img/avatars/2.jpg"></label>

</div>

<div>

<input type="radio" id="A15" name="avatars" value="15">

<label for="A15"><img src="/img/avatars/15.jpg"></label>

</div>

</fieldset>

</p>

<p>

<label for="username" class="sr-only">Username</label>

<input type="text"

th:value="${entity.username}" id="username"name="username" class="form-control" placeholder="Username" required>

</p>

<p>

<label for="password" class="sr-only">Password</label>

<input type="password" id="password" name="password" class="form-control" placeholder="Password" required>

</p>

<p>

<label for="email" class="sr-only">Email</label>

<input type="email" th:value="${entity.email}" ="email" name="email" class="form-control" placeholder="Email" required>

</p>

<p>

<label for="birthDate" class="sr-only">Birthdate</label>

<input type="date" th:value="${entity.birthDate}" id="birthDate" name="birthDate" class="form-control" placeholder="Birth date" required>

</p>

<input hidden name="id" th:value="${entity.id}">

<button class="btn btn-lg btn-primary btn-block" type="submit">Update</button>

</form>

</div>

В итоге мы получаем следующие классы:

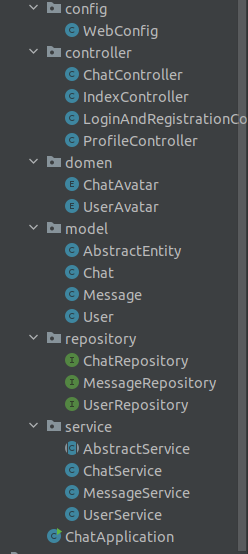


Рисунок 4.1 – Классы приложения

## **Разработка интерфейса программного продукта**

Стартовое окно выглядит следующим образом:

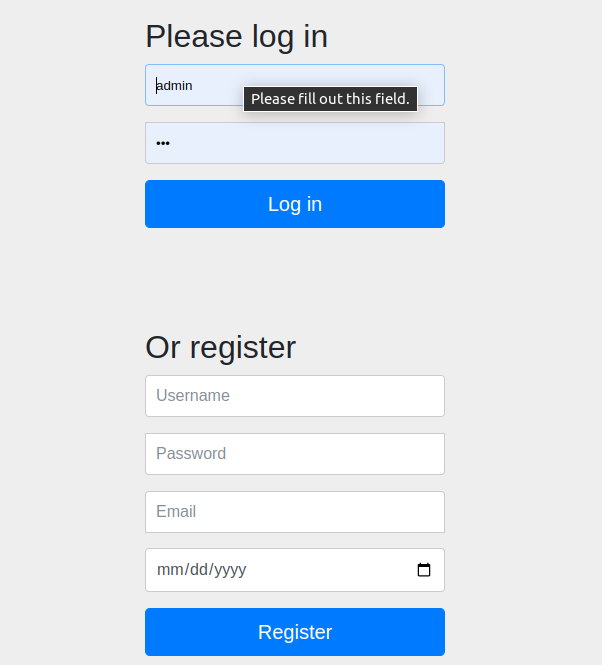


Рисунок 4.2 – Главная страница

После регистрации и входа в систему мы видим список всех чатов.

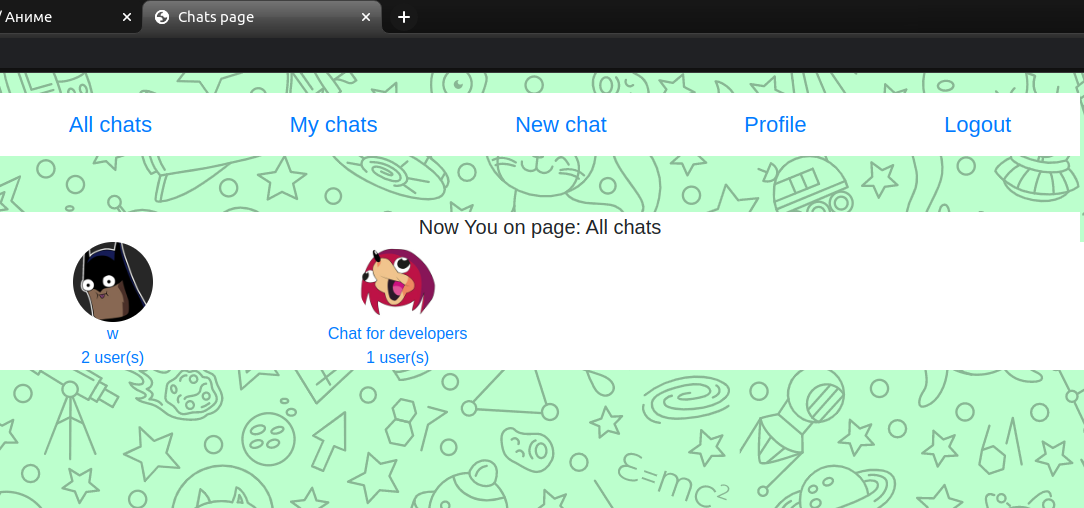


Рисунок 4.3 – Страница списка чатов

Также можно перейти нс страницу чатов которые создал данный пользователь или в которых он участвует.

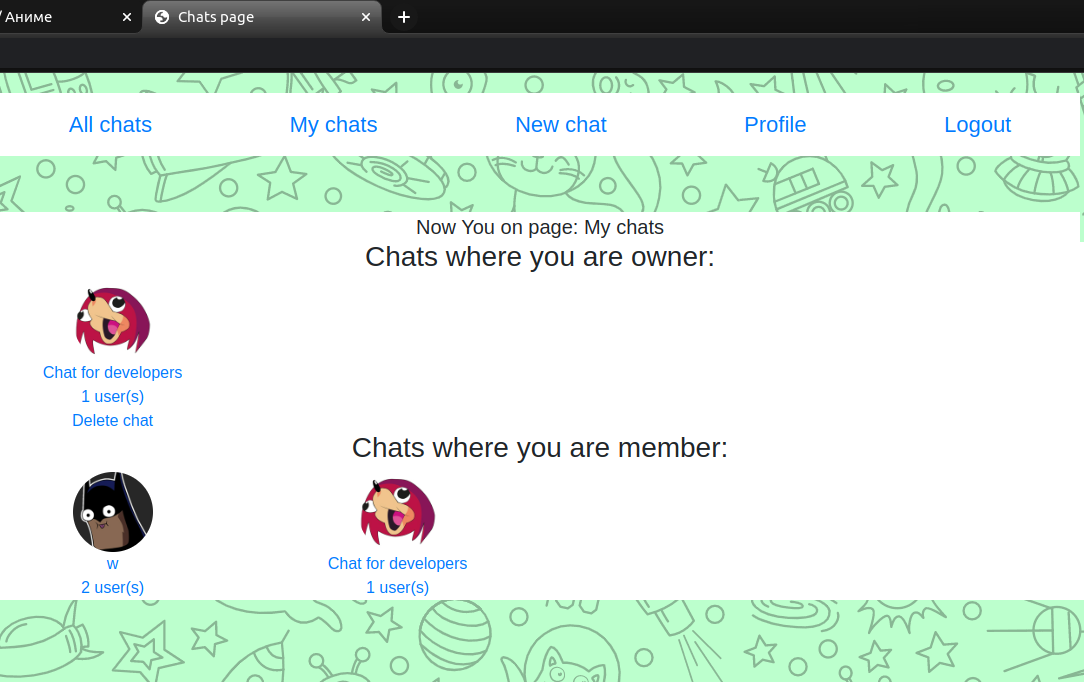


Рисунок 4.4 – Страница моих чатов

Ниже представлен вид страницы открытого чата.

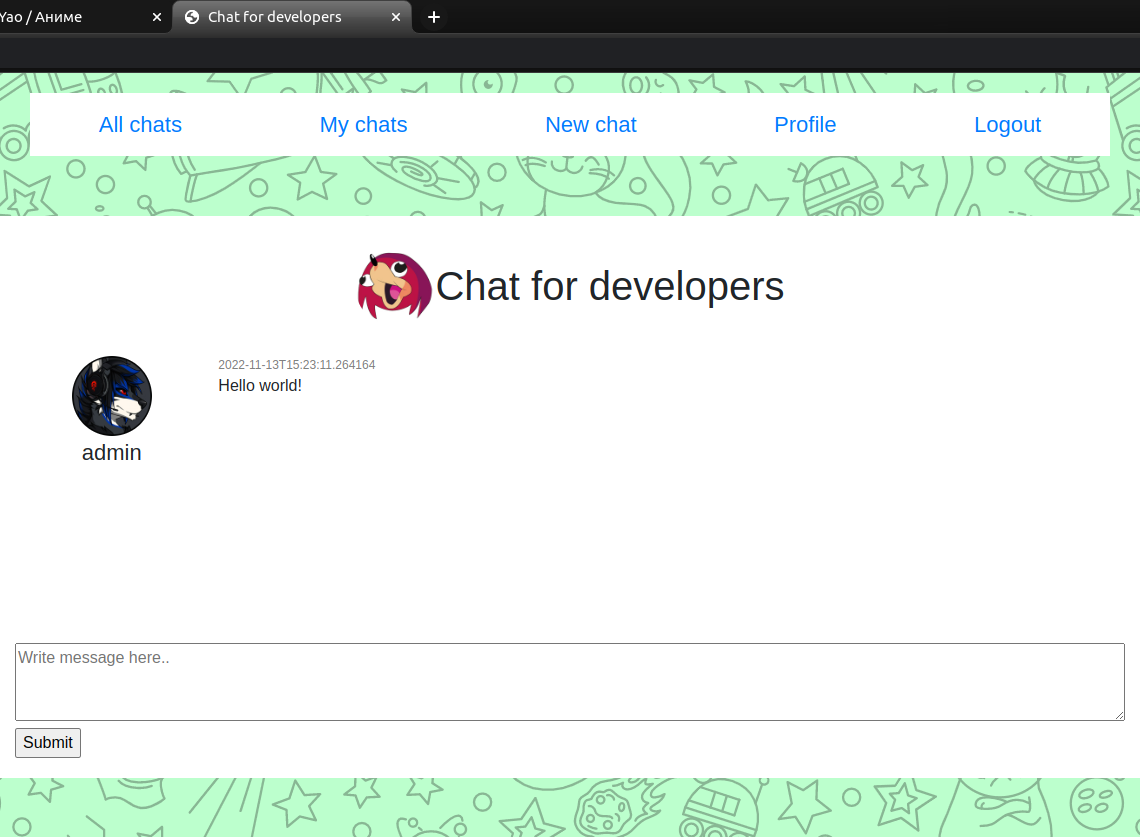


Рисунок 4.5 – Страница чата

Также есть страница профиля, на которой пользователь может обновить информацию о себе и сменить аватарку.

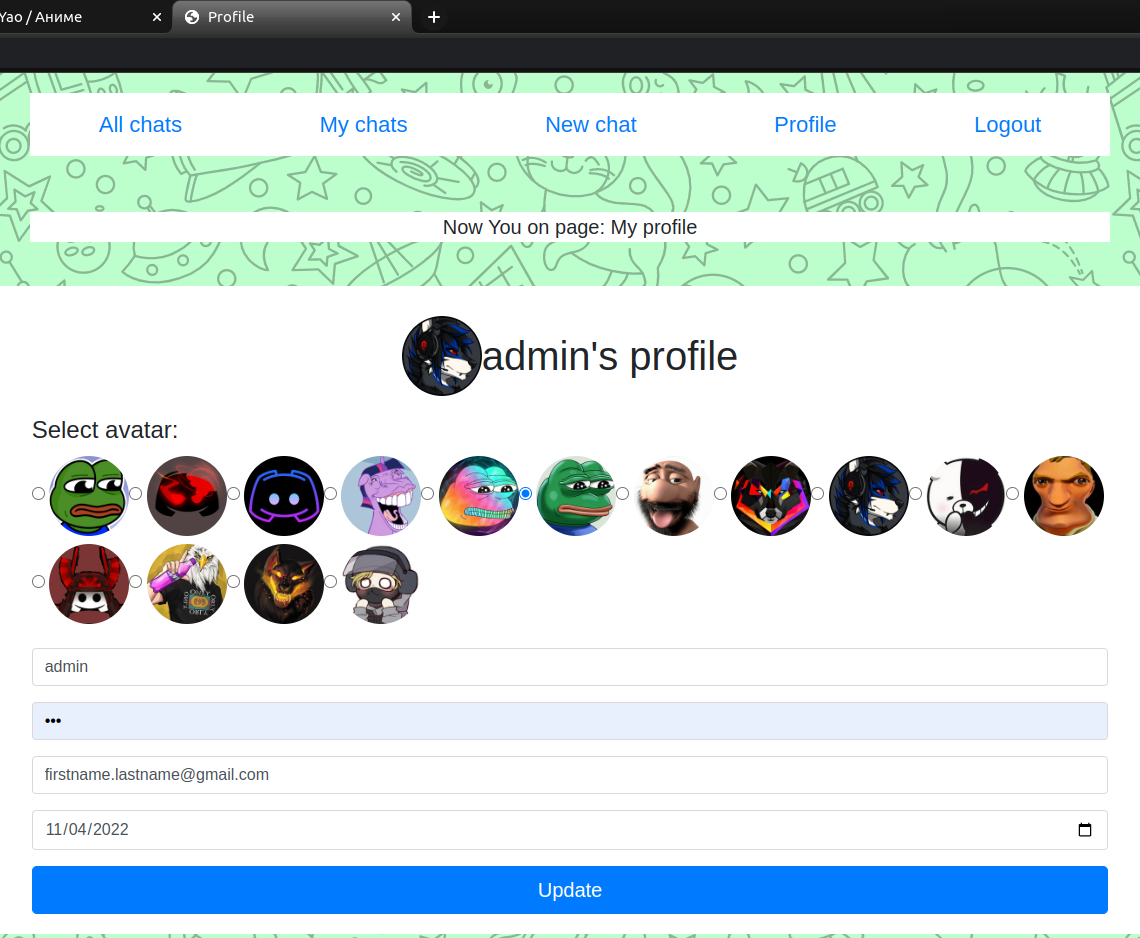


Рисунок 4.6 – Страница профиля

**Разработка алгоритмов реализации вариантов использования.**

Во время разработки приложения было решено множество задач. Решение некоторых из них представим ниже.

1. Одинаковые логины

Нужно было запретить пользователям использовать один и тот же логин для регистрации в системе. Для этого ввели проверку при регистрации

@PostMapping("/registration")

public String registerUser(@ModelAttribute User user,

Model model) {

Optional<User> byUsername = userService.findByUsername(user.getUsername());

if (byUsername.isPresent()) {

model.addAttribute("message", "User with same username already exists");

return "/login.html";

} else {

userService.create(user);

}

return "redirect:/";

}

1. Real time messaging

Необходимо было решить проблему доставки сообщений в реальном времени. Поскольку страницу мы не обновляем, пришлось через js отправлять запросы на обновление сообщений и вручную добавлять новые сообщения на страницу

<script>

function ajaxGET () {

var data = new URLSearchParams();

var nodes = document.querySelectorAll('.messageItem > .hidden');

var last = nodes[nodes.length- 1];

if(!last){

message = 0;

}else{

message = last.textContent;

}

data.append("chat", chatId);

data.append("message", message);

var url = "/chats/messages?" + data.toString();

// (B2) AJAX

var xhr = new XMLHttpRequest();

xhr.open("GET", url);

xhr.onload = function () {

const answer = JSON.parse(this.response);

var newMessage = '';

for(let i=0; i<answer.length; i++){

const msg=answer[i];

newMessage = newMessage +

`<div class="row messageItem">

<div class="hidden id">`+msg['id']+`</div>

<div class="col-2 messageAvatar">

<img src="/img/avatars/`+msg['owner']['avatar'].replace('A','') + `.jpg">

<div class="messageOwner">`+msg['owner']['username']+`</div>

</div>

<div class="col-10">

<div class="dateCreation">`+msg['dateCreation']+`</div>

<div>`+msg['text']+`</div>

</div>

</div>`;

}

const messages = document.querySelector(".messages");

messages.insertAdjacentHTML("beforeend", newMessage);

};

xhr.send();

setTimeout(ajaxGET, 3000);

}

ajaxGET();

</script>

# 5. Тестирование

Проведем тестирование программы, пройдя все варианты взаимодействия. Запустим само приложение. Запуск происходит без проблем.

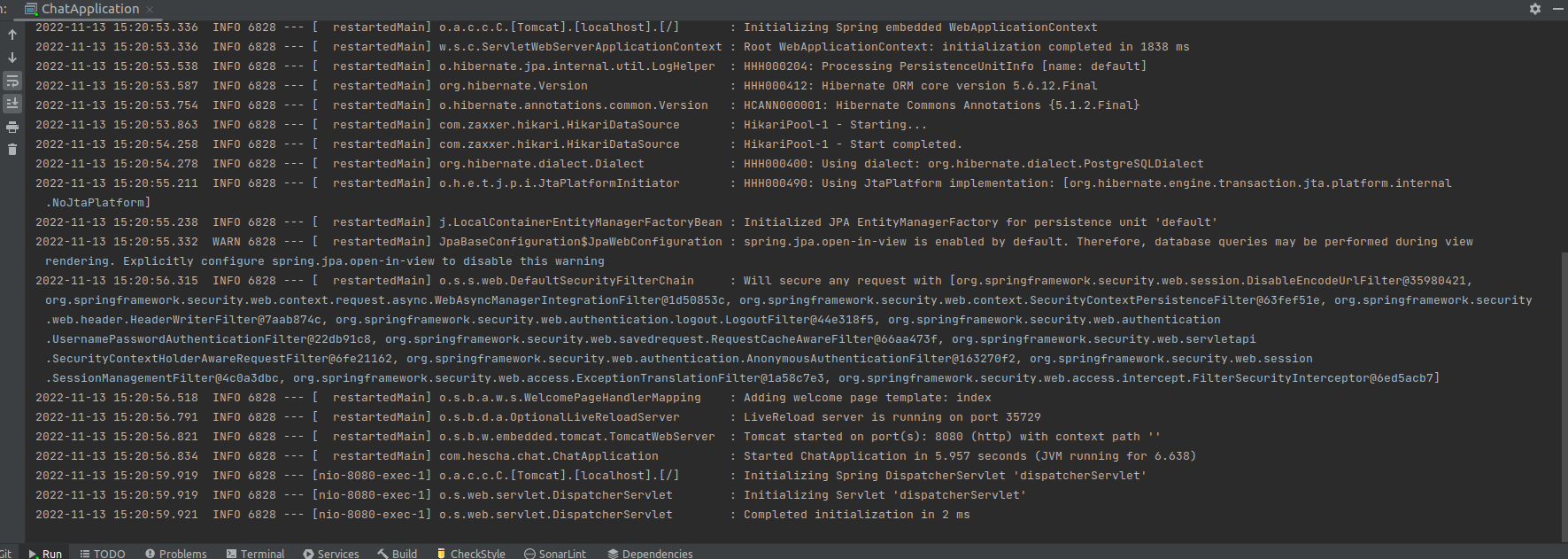


Рисунок 5.1 – Запуск приложения

Зарегистрируемся в системе и войдем пользователем.

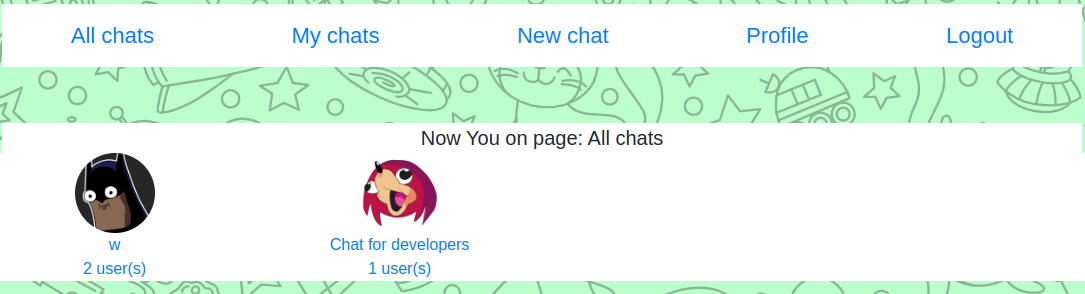


Рисунок 5.2 – Вход пользователем

Создадим свой чат.

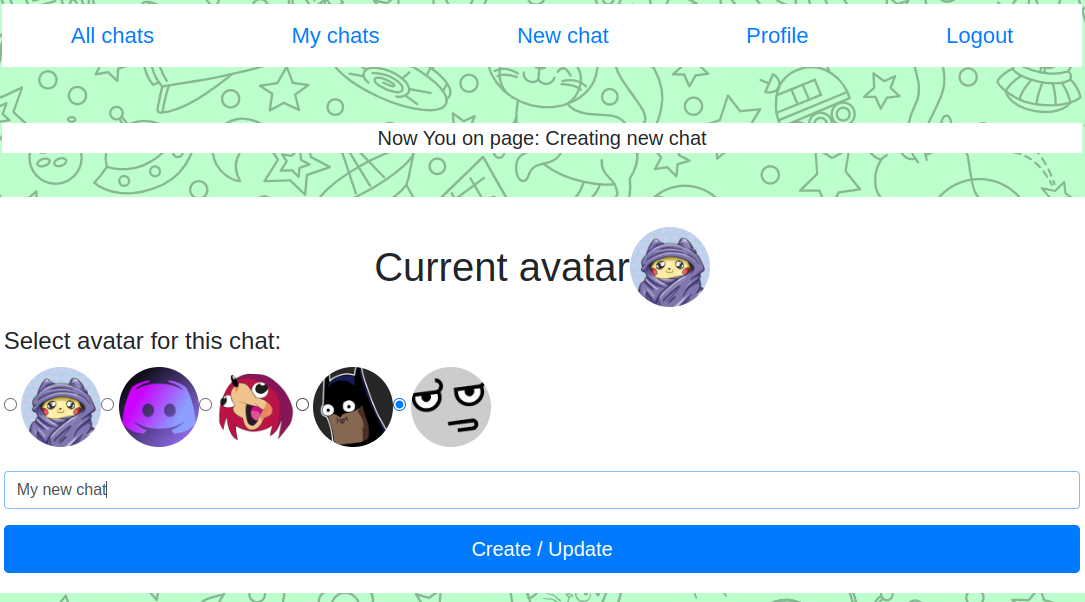


Рисунок 5.3 – Окно создание чата

Напишем сообщение в свой чат.

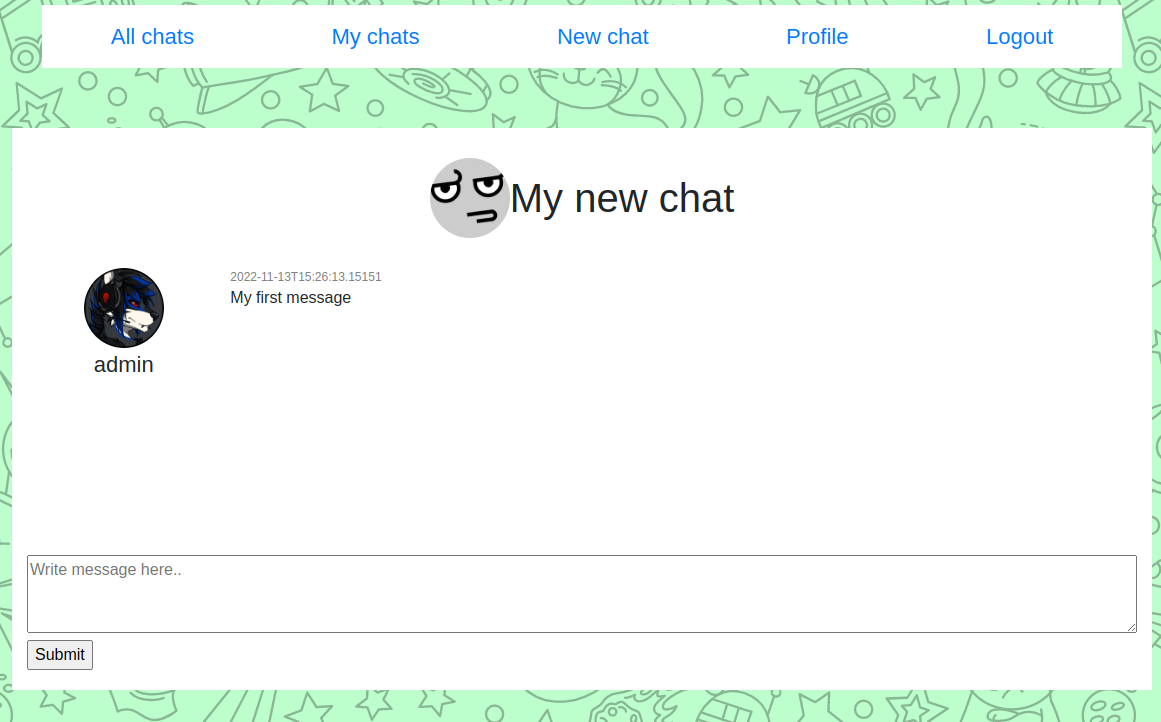


Рисунок 5.4 – Окно чата с собщением

Создадим нового пользователя в другой вкладке браузера и напишем новое сообщение.

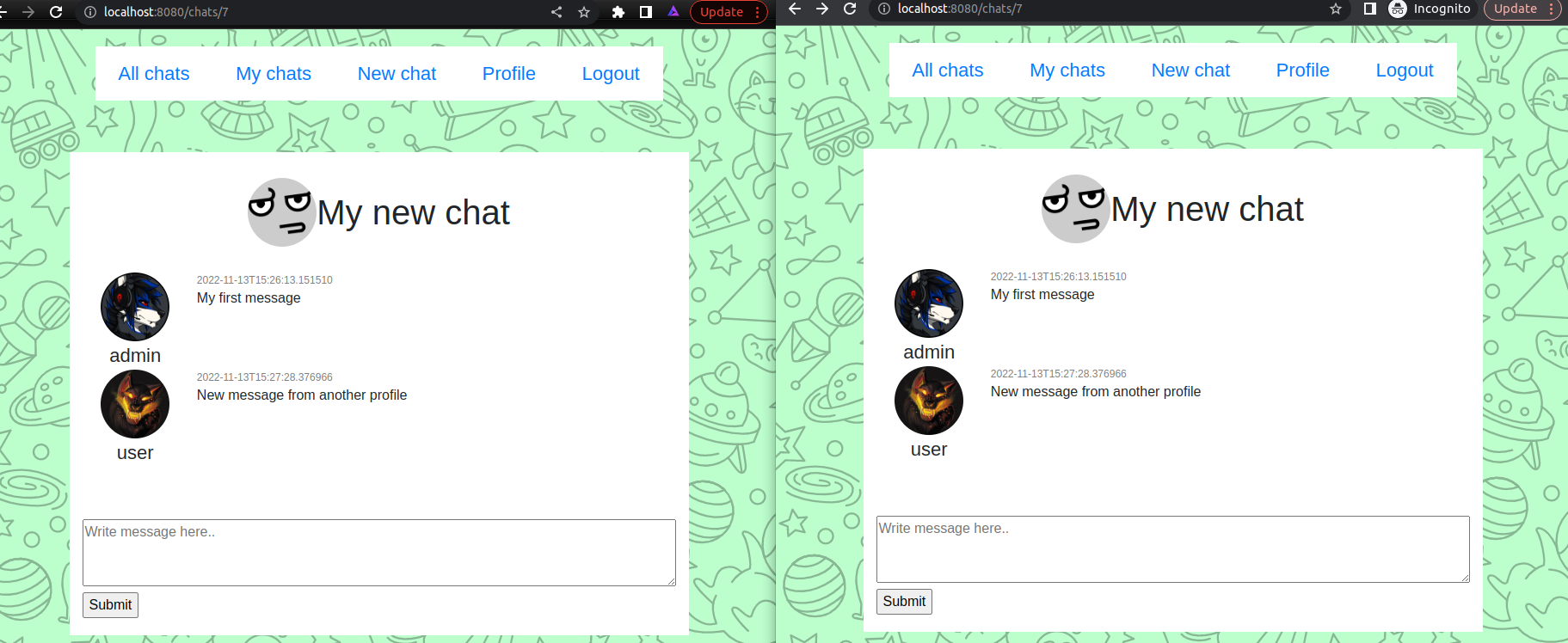
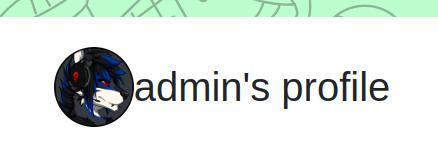


Рисунок 5.5 – Новое сообщение в чате

Зайдем в профиль и обновим аватарку.



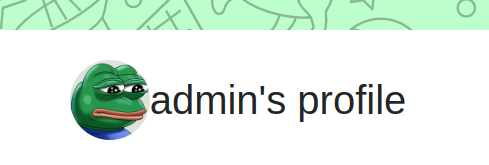


Рисунок 5.6 –Страница профиля

Перейдем в наш чат и увидим что все наши сообщения отображаются с новой аватаркой

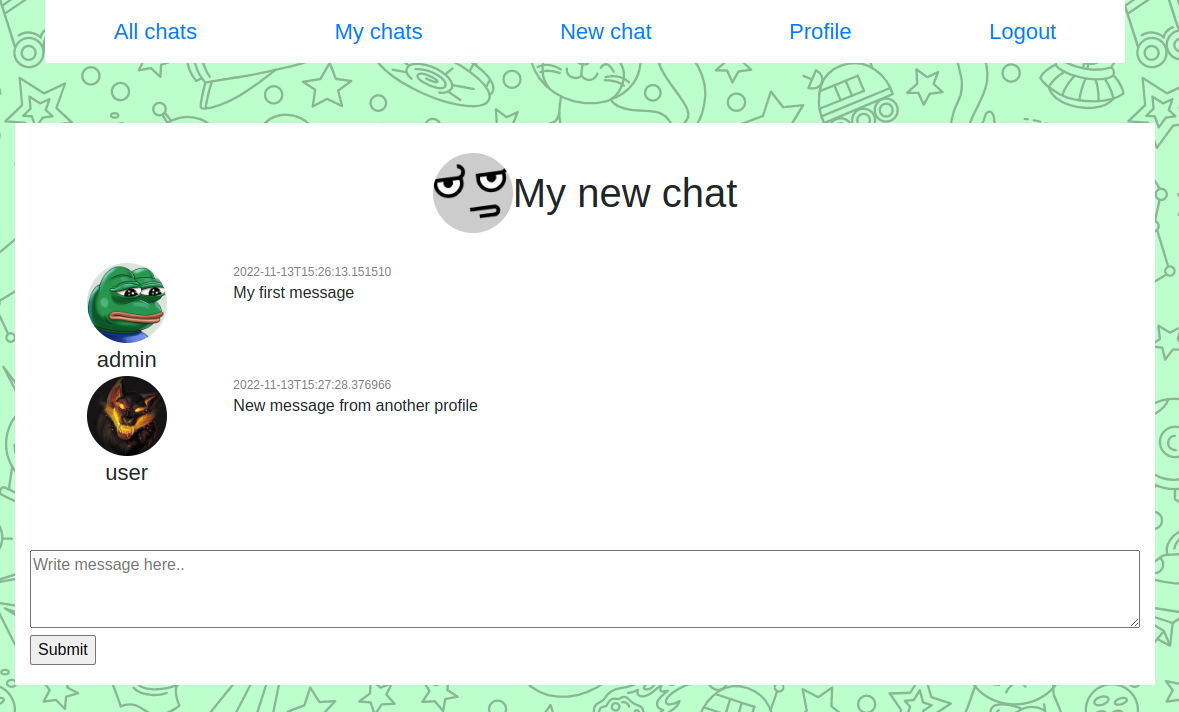


Рисунок 5.7 – Страница чата с обновленной аватаркой

Работа приложения происходит без ошибок.

# Литература

1. Хорстманн, К.С. Java 2. Библиотека профессионала, том1. Основы. 8-е издание, : Пер. с англ. / К. С. Хорстманн, Г. Корнелл - М.: ООО «Вильямс», 2012. - 816 с.
2. Дирк, Л. Самоучитель Java 7: Пер. с нем. / Л. Дирк, П. Мюллер. – СПб: БХВ-Петербург, 2013. – 464 с.: илл.
3. Шилдт, Г. Java. Полное руководство. 8-е издание, : Пер. с англ. / Г. Шилдт. – М.:ООО «Вильямс», 2012. – 1104 с.
4. Блинов, И.Н. Java. Промышленное программирование./ И.Н. Блинов, В.С. Романчик –Минск: «Четыре четверти», 2013. – 896 с.
5. Чат [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%82. – Дата доступа: 13.10.2022.
6. Что такое чат [Электронный ресурс]. – Режим доступа: https://spravkapc.ru/seti-i-internet/chto-takoe-chat-i-kak-im-polzovatsia.html. – Дата доступа: 13.10.2022.
7. Spring Boot [Электронный ресурс]. – Режим доступа: https://spring.io/projects/spring-boot. – Дата доступа: 13.10.2022.
8. HTML tutorial [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/html/. – Дата доступа: 13.10.2022.
9. CSS tutorial [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/css/. – Дата доступа: 13.10.2022.
10. Learn JS [Электронный ресурс]. – Режим доступа: https://learn.javascript.ru/. – Дата доступа: 13.10.2022.
11. IntelijIdea [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/idea/. – Дата доступа: 13.10.2022.

## Приложение А

Исходный код, откомпилированная программа и записка находятся на диске.