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

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практическая РАБОТа №4

по дисциплине «Архитектура клиент-серверных приложений»

Выполнил студент группы ИКБО-13-19 Малютина В.А.

Принял Степанов П.В.

Практическая работа выполнена «\_\_\_»\_\_\_\_\_\_\_2021г. *подпись студента*

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021г. *подпись преподавателя*

Москва 2021

**Содержание**

[Цель работы 3](#_Toc90465097)

[Задание 3](#_Toc90465098)

[Теоретическое введение 3](#_Toc90465099)

[Выполнение работы 4](#_Toc90465100)

[ЗАКЛЮЧЕНИЕ 10](#_Toc90465101)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc90465102)

# Цель работы

Ознакомить**ся** с многослойными клиент-серверными архитектурами, посмотреть разницу между ними, выявить плюсы и минусы.

# Задание

Поскольку для трёхуровневой архитектуры необходимо физическоеразделение подсистем, то предлагается разработать трёхслойноеприложение: БД, сервер, приложение. В качестве приложения можноиспользовать: запросы вpostman/insomnia/testmace, простой сайт,десктопное приложение, мобильное приложение. В качестве БД можноиспользовать: SQLite, PostgreSQL.

# Теоретическое введение

**Тонкие клиенты**

Тонкий клиент спроектирован так, чтобы основная часть обработки данных происходила на сервере. Тонкий клиент как правило без жесткого диска: действуют как простой терминал к серверу и требует постоянной связи с сервером.

**Толстые клиенты**

Толстый клиент выполняет основную часть обработки. У толстых клиентов нет необходимости в непрерывной связи с сервером, поскольку они в основном передают информацию на сервер.

**Когда какой клиент использовать**

Тонкие клиенты обеспечивают работу рабочего стола в средах, где конечный пользователь имеет четко определенное и регулярное количество задач, для которых используется система. Тонких клиентов можно найти в медицинских офисах, авиабилетах, школах, правительствах, производственных предприятиях и даже колл-центрах. Наряду с простотой установки, тонкие клиенты также предлагают более низкую общую стоимость владения по сравнению с толстыми клиентами. Если вашим приложениям требуются мультимедийные компоненты или которые интенсивно используют пропускную способность, стоит присмотреться к разработке толстых клиентов. Одно из самых больших преимуществ толстых клиентов – некоторые операционные системы и программное обеспечение не могут работать на тонких клиентах. Толстые клиенты могут справиться с ними, поскольку у них есть свои собственные ресурсы.

# Выполнение работы

В данной практической работе был реализован чат на веб-сокетах. Для запуска базы данных использовался Docker. Использована система сборка Maven и использован паттерн проектирования MVC. В проекте представлены такие классы (Рисунок 1):

* 3 класса конфигураций
* 2 контроллера, Main и Registration (отвечающий за регистрацию)
* 4 сущности, основные это сущность Пользователь и Сообщение
* 2 репозитория для сущностей Сообщение и Пользователь
* 1 сервис

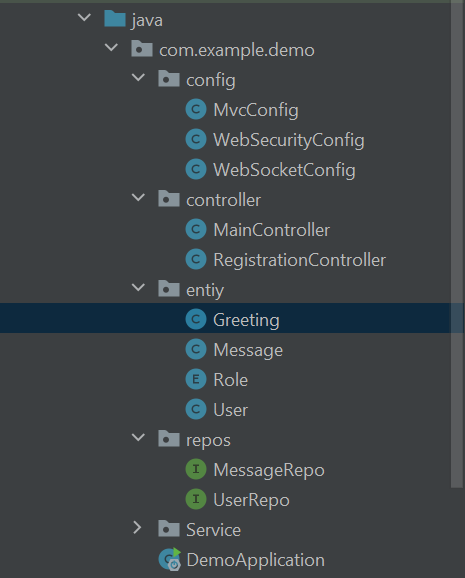


Рисунок 1 – Структура классов

Далее будут показаны классы с конфигурациями (Листинг 1-3).

Листинг 1 – MvcConfig

package com.example.demo.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.core.session.SessionRegistry;  
import org.springframework.security.core.session.SessionRegistryImpl;  
import org.springframework.security.web.session.HttpSessionEventPublisher;  
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
@Configuration  
public class MvcConfig implements WebMvcConfigurer {  
  
 public void addViewControllers(ViewControllerRegistry registry) {  
 registry.addViewController("/login").setViewName("login");  
 }  
 @Bean  
 public HttpSessionEventPublisher httpSessionEventPublisher(){  
 return new HttpSessionEventPublisher();  
 }  
  
}

Листинг 2 – WebSecurityConfig

package com.example.demo.config;  
  
import com.example.demo.Service.UserServ;  
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.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.builders.WebSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.core.session.SessionRegistry;  
import org.springframework.security.core.session.SessionRegistryImpl;  
import org.springframework.security.crypto.password.NoOpPasswordEncoder;  
import org.springframework.security.web.util.matcher.AndRequestMatcher;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
  
@Configuration  
@EnableWebSecurity  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
 @Autowired  
 private UserServ userServ;  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .authorizeRequests()  
 .antMatchers("/reg").permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .loginPage("/login")  
 .permitAll()  
 .and()  
 .logout().deleteCookies("JSESSIONID").logoutRequestMatcher(new AntPathRequestMatcher("/logout"))  
 .permitAll()  
 .and()  
 .rememberMe().key("uniqueAndSecret").tokenValiditySeconds(86400)  
 .and()  
 .sessionManagement()  
 .maximumSessions(-1)  
 .sessionRegistry(sessionRegistry());  
 http.csrf().disable();  
 }  
 @Override  
 public void configure(WebSecurity web) { //подзагрузка css, картинки  
 web.ignoring().antMatchers("/style/\*\*","/error","/js/\*\*","/img/\*\*");  
 }  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception{  
 auth.userDetailsService(userServ)  
 .passwordEncoder(NoOpPasswordEncoder.*getInstance*());  
 }  
 @Bean  
 SessionRegistry sessionRegistry(){  
 return new SessionRegistryImpl();  
 }  
}

Листинг 3 – WebSocketConfig

package com.example.demo.config;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.messaging.simp.config.MessageBrokerRegistry;  
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;  
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;  
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;  
  
@Configuration  
@EnableWebSocketMessageBroker  
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {  
  
 @Override  
 public void configureMessageBroker(MessageBrokerRegistry config) {  
 config.enableSimpleBroker("/topic");  
 config.setApplicationDestinationPrefixes("/app");  
 }  
  
 @Override  
 public void registerStompEndpoints(StompEndpointRegistry registry) {  
 registry.addEndpoint("/webs").withSockJS();  
 }  
  
}

Классы контроллеры (Листинг 4-5)

Листинг 4 – MainController

@Controller  
public class MainController {  
 @Autowired  
 private UserRepo userRepo;  
 @Autowired  
 private MessageRepo messageRepo;  
 @GetMapping("/")  
 public String getChat(@AuthenticationPrincipal User user, Model model){  
 User userBD = userRepo.findById(user.getId()).orElse(new User());  
 List<Message> messageBDList=messageRepo.findAll();  
 model.addAttribute("messages",messageBDList);  
 model.addAttribute("username",userBD.getUsername());  
 return "chat";  
 }  
 @MessageMapping("/webs")  
 @SendTo("/topic/webs")  
 public Message greeting(Message message) throws Exception {  
 message.setTime(new SimpleDateFormat("yyyy.MM.dd:HH.mm.ss").format(Calendar.*getInstance*().getTime()));  
 messageRepo.save(message);  
 Thread.*sleep*(1000);  
 return message;  
 }  
}

Листинг 5 – RegisrationController

@Controller  
public class RegistrationController {  
 @Autowired  
 private UserRepo userRepo;  
 @GetMapping("/reg")  
 public String regPage(){  
 return "reg";  
 }  
 @PostMapping("/reg")  
 public String addUser(User user, Model model) {  
 UserDetails userBD = userRepo.findByUsername(user.getUsername());  
 if (userBD != null) {  
 model.addAttribute("message", "User exist!");  
 return "reg";  
 }  
 user.setActive(true);  
 user.setRoles(Collections.*singleton*(Role.*USER*));  
 userRepo.save(user);  
 return "redirect:/login";  
 }  
}

Все сущности показаны не будут, более подробно можно будет посмотреть на GitHub, далее продемонстрированы сущности User и Message (Листинг 6-7).

Листинг 6 – User

@Entity  
@Table(name = "users")  
@Data  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String username;  
 private String password;  
 private boolean active;  
 @ElementCollection(targetClass = Role.class, fetch = FetchType.*EAGER*)  
 @CollectionTable(name = "user\_role",joinColumns = @JoinColumn(name = "user\_id"))  
 @Enumerated(EnumType.*STRING*)  
 private Set<Role> roles;  
 private String email;  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return roles;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

Листинг 7 – Message

@Data  
@Entity  
public class Message {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String author;  
 private String content;  
 private String time;  
}

В заключение проект был развернут на Heroku (Рисунок 2-3).

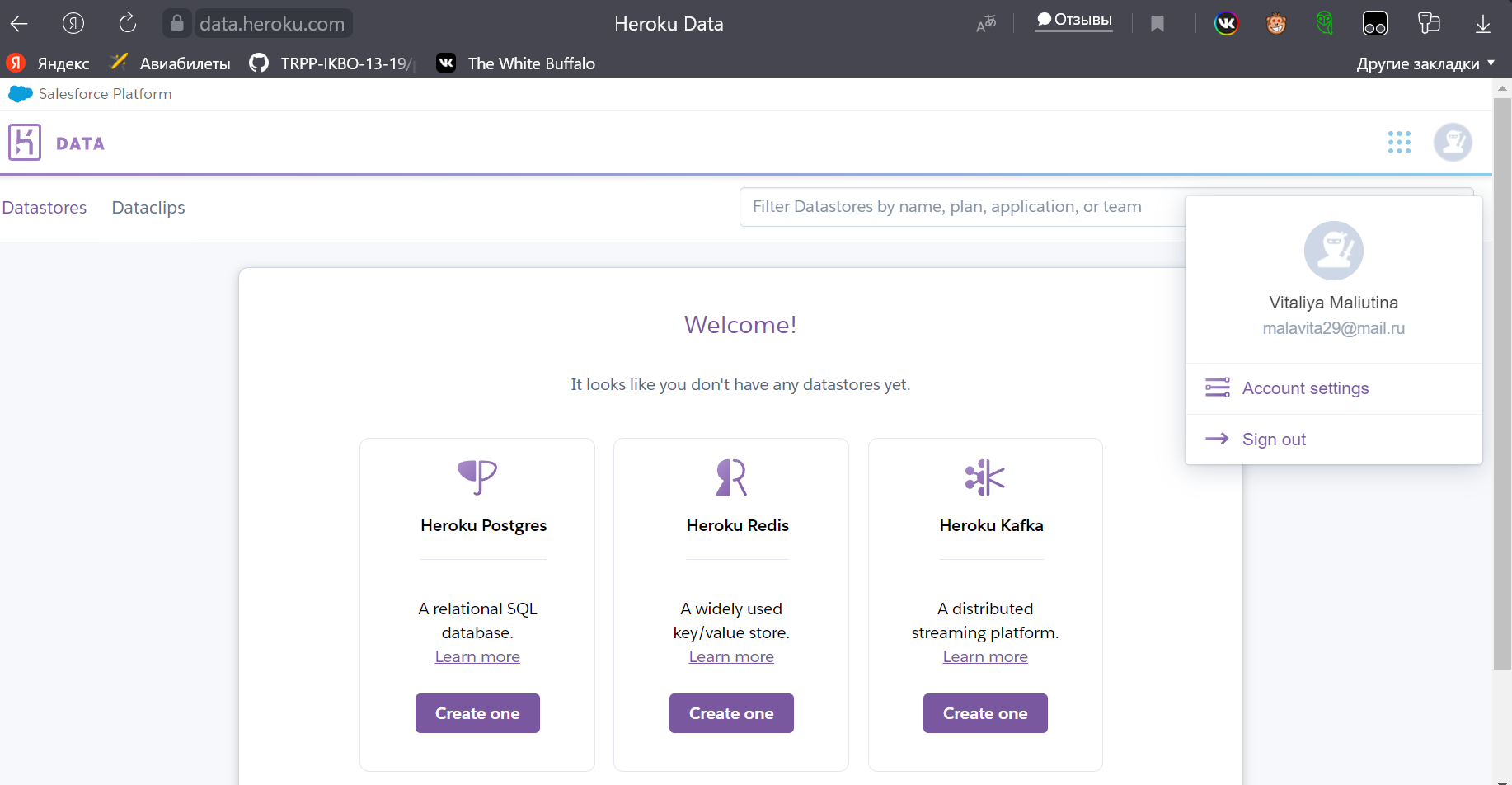


Рисунок 2 – Heroku.com

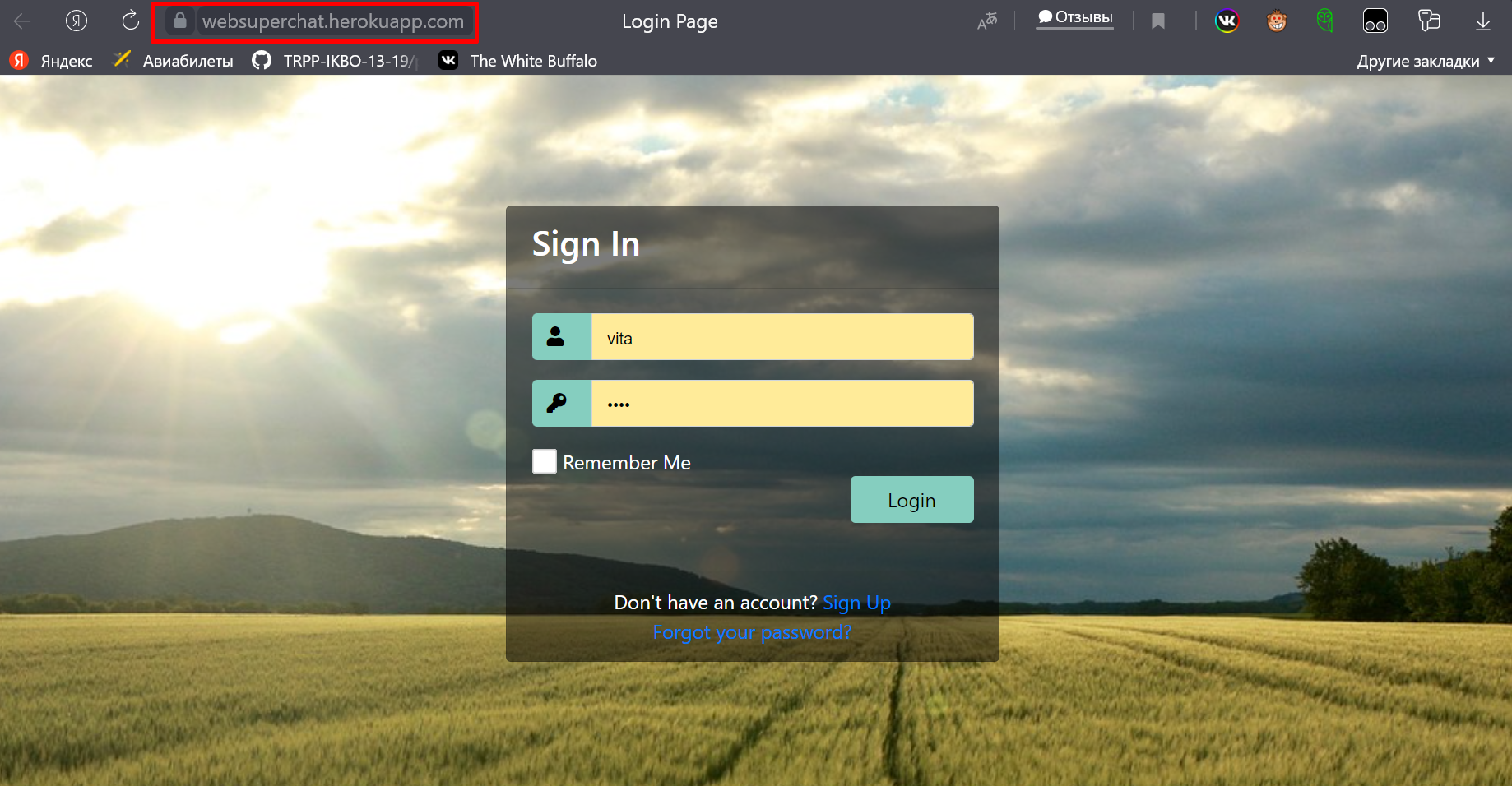


Рисунок 3 – Развернутое приложение на Heroku

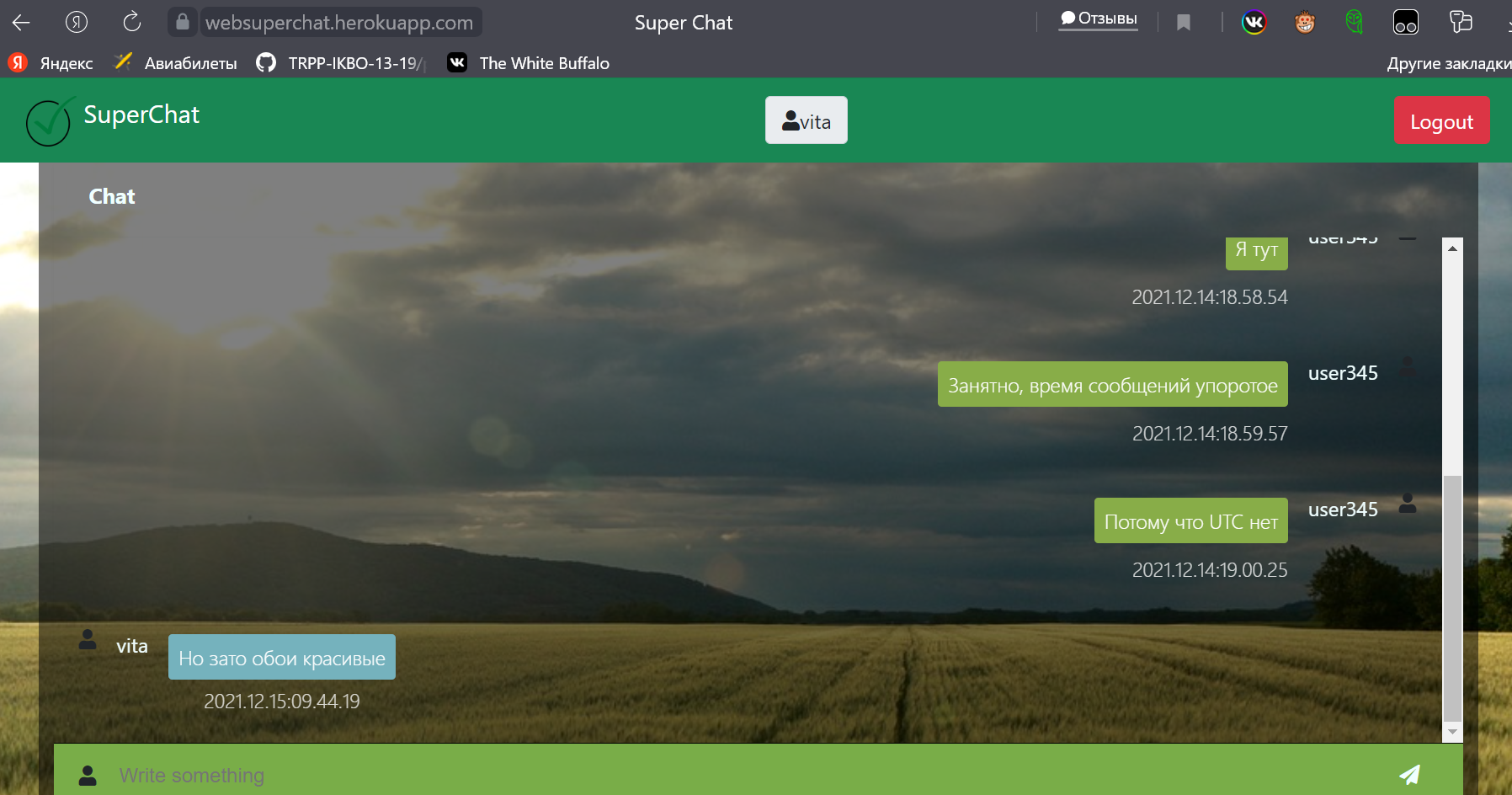


Рисунок 4 – Чат

# ЗАКЛЮЧЕНИЕ

В результате выполнения практической работы мы ознакомились с многослойными клиент-серверными архитектурами. Был создан чат на веб-сокет. Приложение было развернуто на Heroku.com.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алпатов А.Н. Архитектура клиент-серверных приложений: конспект лекций. РТУ МИРЭА, Москва, 2021.
2. Методическое пособие к практической работе №5 по дисциплине «Архитектура клиент-серверных приложений», РТУ МИРЭА, Москва, 2021.
3. Sysout.ru [Электронный ресурс] / Чат на Spring Boot и WebSocket – режим доступа: https://sysout.ru/chat-na-spring-boot-i-websocket/. – Загл. с экрана. – Яз. рус.