Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет Информационных технологий и радиоэлектроники

Кафедра Информационных систем

КУРСОВАЯ

РАБОТА

по курсу Прикладная разработка на Java

на тему: Веб-приложение "Социальная сеть"

Руководитель

к. т. н., доц. каф. ИС

(уч. степень, звание)

Метелкин А.С.

(оценка) (фамилия, инициалы)

(подпись) (дата)

Члены комиссии Студент ИС - 122 (группа)

Крюков. Д. Н.

(подпись) (Ф.И.О.) (фамилия, инициалы)

(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2025

В курсовой работе отражено создание веб-приложения "Социальная сеть",

разработано веб-приложение с интуитивным управлением;

проведена отладка и тестирование приложения для различных незапланированных действий.

Для работы было использовано 4 таблицы базы данных: User, Message, Conversation и MidConversation, а так же было использовано около 10 зависимостей Maven.

В работе, в отделе разработки, указана 1 таблица и 21 иллюстрация.

Приведены примеры работы программы.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#_Toc105757734) 4

[1 Анализ технического задания](#_Toc105757735) 5

[1.1 Общее описание системы](#_Toc105757741) 5

[1.2 Технические требования](#_Toc105757736) 5

[1.3 Функциональные требования](#_Toc105757737) 6

[1.4 Нефункциональные требования](#_Toc105757737) 6

[1.5 Архитектура приложения](#_Toc105757737) 6

[1.6 Сравнение технологий](#_Toc105757737) 7

[1.7 Процесс разработки](#_Toc105757737) 7

[1.8 Ожидаемые результаты](#_Toc105757737) 8

[1.9 Выбор технологий и инструментов](#_Toc105757737) 8

[2 Проектирование программы](#_Toc105757739) 9

2.1 [Проектирование структуры базы данных](#_Toc105757739) 9

2.2 [Проектирование REST API](#_Toc105757739) 10

2.3 [Архитектура клиент-сервисного взаимодействия](#_Toc105757739) 12

2.4 [Паттерны проектирования в разработке системы](#_Toc105757739) 13

[3 Разработка программы](#_Toc105757740)…………………………………………………….....….16

3.1 [Реализация бэкенда](#_Toc105757739) 16

3.2 [Разработка фронтенда](#_Toc105757739) 23

[4 Тестирование](#_Toc105757742) ………………………………………………………………...……25

4.1 Unt-тесты.……....……………………………………...….……………….…….25

4.2 Проверка функциональности веб-приложения..…………….……….……….27

[ЗАКЛЮЧЕНИЕ](#_Toc105757744) ..39

[СПИСОК ЛИТЕРАТУРЫ](#_Toc105757744) 40

**ВЕДЕНИЕ**

В последние десятилетие социальные сети стали неотъемлемой частью жизни миллионов людей по всему миру. Они предоставляют платформу для общения, обмена информацией, а также для создания сообществ, объединенных общими интересами. Социальные сети влияют на различные аспекты повседневной жизни, включая бизнес, образование, культуру и даже политику. В связи с этим, разработка эффективных и безопасных веб-приложений для социальной сети является актуальной задачей.

Целью данной курсовой работы является разработка веб-приложения социальной сети с использованием фреймворка Spring Boot. В процессе разработки будет реализован функционал для создания профилей пользователей, аутентификации, обмена сообщениями, формировании истории сообщений, создания бесед и поиска пользователей. Особое внимание будет уделено обеспечению масштабируемости приложения.

Веб-приложение будет построено с использованием современных технологий и инструментов, таких как Spring Boot для серверной части, Hibernate для работы с базой данных, а также frontend-технологий для взаимодействия с пользователем. Одним из приоритетов разработки будет обеспечение простоты использования и быстродействия интерфейса, а также создание надежной архитектуры, способной выдерживать высокие нагрузки.

Основная цель проекта: Предоставить возможность пользователям интернета общаться между собой, создавать беседы и искать новых собеседников.

В ходе работы также будет проведено тестирование функциональности и производительности системы для оценки её готовности к реальной эксплуатации.

**1 Анализ технического задания**

По заданию было созданно веб-приложение «Социальная сеть», которое копирует часть основного функционала из других социальных сетей. В качестве языка программирования был использован Java, а среда разработки – мощная интегрированная среда разработки (IDE) от JetBrains IntelliJ IDEA Ultimate, со сборщиком проектов и инструментом управления зависимости Maven.

**1.1 Общее описание системы**

Ключевые функции:

- регистрация и аутентификация пользователей;

- создание и редактирование профиля;

- личная и групповая переписка;

- добавление пользователей в беседу.

**1.2 Технические требования**

Ключевые требования к системе:

- скорость работы и производительность. Например, приложение

должно поддерживать работу с множеством пользователей одновременно и обеспечивать быстрый отклик при взаимодействии с лентой новостей или отправке сообщений;

- масштабируемость. Как система будет расширяться, если

потребуется поддержка большего числа пользователей и данных (например, распределение нагрузки, использование кеширования);

- надежность и отказоустойчивость. Как система будет реагировать на

сбои, например, база данных будет работать в режиме репликации или с использованием бэкап-системы.

**1.3 Функциональные требования**

Функциональные возможности системы:

- регистрация и аутентификация, описание процесса регистрации

новых пользователей, включая проверку email, пароль;

- профили пользователей, могут редактировать свои личные данные;

- поиск пользователей и история переписок, пользователи могут друг

друга найти по email и история их переписок будет всегда доступна для пользователя;

- отправка сообщений и создание бесед.

Пользователи всегда смогут написать друг другу сообщение и создать

беседу на несколько пользователей сразу.

**1.4 Нефункциональные требования**

Это дополнительные требования, которые не связаны напрямую с функциональностью, но важны для качественной работы системы:

- юзабилити. Простота в использовании, интуитивно понятный интерфейс;

- отказоустойчивость. Как приложение будет восстанавливать

состояние в случае сбоя.

**1.5 Архитектура приложения**

Основные компоненты системы и их взаимодействие:

- backend: использование Spring Boot для серверной части, работа с

базой данных через Hibernate (JPA), организация REST API для взаимодействия с frontend;

- frontend: использование фреймворков или библиотек для создания

интерфейса через Thymeleaf с HTML;

- база данных: описание структуры БД, использование реляционной базы данных PostgreSQL, основные таблицы: Users, Messages, Conversations, Midconversations.

**1.6 Сравнение технологий**

Для реализации веб-приложения был выбран стек технологий: Spring Boot для разработки серверной части, Thymeleaf для рендеринга HTML-страниц и PostgreSQL в качестве базы данных. В процессе выбора и анализа других технологий были рассмотрены альтернативы.

Spring Boot vs. Java EE (Jakarta EE): Несмотря на существование Java EE, Spring Boot был выбран за его простоту, гибкость и богатую экосистему. Он предлагает значительно меньше настроек "из коробки" и позволяет быстрее разрабатывать приложение.

Thymeleaf vs. JSP: В отличие от традиционного JSP, Thymeleaf позволяет создавать более чистый и легко поддерживаемый код, так как шаблоны Thymeleaf являются валидным HTML. Это позволяет легко интегрировать серверную часть с фронтендом и избегать множества проблем с форматированием.

PostgreSQL vs. MySQL: PostgreSQL был выбран за его поддержку сложных запросов и транзакционную надежность, что особенно важно для работы с большими объемами данных и обеспечения целостности данных.

В результате, выбранный стек технологий оказался наиболее подходящим для создания масштабируемого и надежного веб-приложения.

**1.7 Процесс разработки**

Описание процесса реализации приложения:

- анализ требований и проектирование архитектуры;

- проектирование базы данных и создание схемы таблиц;

- разработка API для взаимодействия между клиентом и сервером;

- реализация функционала: от регистрации пользователей до поиска

пользователей и отправки сообщений;

- тестирование: описание методов тестирования, таких как юнит

тесты, интеграционные тесты для проверки работы API.

**1.8 Ожидаемые результаты**

Разработанное веб-приложение социальной сети с функциональностью, аналогичной популярным социальным платформам. Удовлетворение всех функциональных и нефункциональных требований. Применение лучших практик безопасности и производительности.

**1.9 Выбор технологий и инструментов**

Для работы был выбран следующий перечень инструментов:

- spring boot: за счет упрощения конфигурации и настройки

приложения, хорошей поддержкой работы с базой данных, безопасности и удобства масштабирования;

- hibernate/JPA: для упрощения работы с базой данных,

автоматической генерации SQL-запросов и работы с объектами;

- postgreSQL: выбор реляционной базы данных для хранения данных;

- frontend: какой будет использоваться шаблонизатор Thymeleaf для

отображения интерфейса и переменных с сервера.

**2. Проектирование программы**

**2.1 Проектирование структуры базы данных**

В данном разделе описываются сущности (таблицы) базы данных и их

атрибуты, используемые в игровом центре. База данных состоит из нескольких

таблиц, каждая из которых отвечает за хранение информации об играх и

результатах сессий.

1. Таблица User

Эта таблица хранит информацию о пользователях.

Атрибуты:

ID – уникальный идентификатор пользователя;

NAME – имя пользователя;

LASTNAME – фамилия пользователя;

DATE\_OF\_BIRCH – дата рождения пользователя;

GENDER – пол пользователя;

СITY – город проживания пользователя;

EMAIL – электронная почта пользователя;

PASSWORD – пароль пользователя;

СOLORSTYLE – установка дизайнерских настроек пользователя.

1. Таблица Message

Эта таблица хранит сообщения между пользователями.

Атрибуты:  
ID – уникальный идентификатор сообщения;

MESSAGE – текст сообщения;

FIRSTID – уникальный идентификатор пользователя, написавший первое сообщение;

SECONDID – уникальный идентификатор пользователя, написавший после первого пользователя.

1. Таблица Conversation

Эта таблица хранит информацию о беседах между пользователями.

Атрибуты:

ID – уникальный идентификатор беседы;

MESSAGE – текст сообщения беседы;

NAMEOFCONVERSATEION – название беседы;

IDOWNER – уникальный идентификатор пользователя, который создал беседу;

ADMINIFOWNER – обозначение для ограничения для добавление в беседу других пользователей.

4. Таблицу MidConversation

Эта таблица хранит списки пользователей с указанием, в какой беседе они состоят.

Атрибуты:

ID – уникальный идентификатор промежуточной таблицы;

IDOFUSER – уникальный идентификатор пользователя, состоящий в беседе;

IDOFCONVERSATEION – уникальный идентификатор для беседы, в которой состоит пользователь.

**2.2 Проектирование REST API**

В проекте реализованы REST API-эндпоинты для регистрации, аутентификации, отправки сообщений и получения данных о пользователях и чатах. Используются аннотации Spring @RestController, @RequestMapping, @PostMapping, @GetMapping, @PathVariable, @RequestBody и др.

Реализация API:

**- Регистрация пользователя**

Эндпоинт: POST /api/v1/registration  
Контроллер: UserController  
Метод принимает JSON с данными нового пользователя и вызывает метод register() в UserService. После регистрации данные сохраняются в базе через UserRepository.

**- Аутентификация по email и паролю**  
Эндпоинт: POST /api/v1/entranceByEmail  
Контроллер: UserController  
Метод login() получает email и пароль, проверяет пользователя через UserService, возвращает JWT или другой маркер доступа.

**- Отправка сообщения другому пользователю**  
Эндпоинт: POST /api/v1/users/{id}/message  
Контроллер: UserController  
Метод принимает ID получателя как @PathVariable и тело сообщения как @RequestBody, вызывает sendMessage() в UserService, где создаётся и сохраняется новое сообщение.

**- Получение списка всех пользователей**  
Эндпоинт: GET /api/v1/users  
Контроллер: UserController  
Возвращает список всех пользователей в виде DTO.

**- Получение чатов пользователя**  
Эндпоинт: POST /api/v1/ conversation /{id}   
Контроллер: UserController  
Получает список чатов, в которых участвует пользователь с заданным ID. Данные берутся через UserService и ConversationRepository.

Особенности реализации

- Используются HTTP-методы в соответствии с назначением: POST — для создания, GET — для чтения.

- Все маршруты сгруппированы по версии API (/api/v1/), что упрощает поддержку и расширение.

- Используются DTO-объекты для передачи данных в JSON-формате между клиентом и сервером.

- Возвращаются стандартные HTTP-коды (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized и др.), обработка ошибок реализована через @ControllerAdvice.

**2.3 Архитектура клиент-сервисного взаимодействия**

REST (Representational State Transfer) — архитектурный стиль построения

распределённых приложений, основанный на использовании стандартных HTTP-методов (GET, POST, PUT, DELETE) для взаимодействия с ресурсами. REST API позволяет клиентам выполнять операции над ресурсами через единую, стандартизированную интерфейсную точку.

Spring Boot — популярный фреймворк для создания приложений на Java, который упрощает разработку REST API за счёт удобной конфигурации, автоконфигурации и готовых шаблонов.

Основные принципы REST API

1. Ресурсы — все объекты, с которыми работает API (например,

пользователи, товары, заказы) идентифицируются уникальными URI.

1. HTTP методы — CRUD операции реализуются через методы:

- GET — получение ресурса или списка ресурсов

- POST — создание нового ресурса

- PUT — обновление существующего ресурса

- DELETE — удаление ресурса

1. Статусы HTTP — сервер возвращает соответствующие коды статуса (200 OK, 201 Created, 404 Not Found и др.)
2. Без состояния (stateless) — каждый запрос содержит всю необходимую

информацию для обработки, сервер не хранит состояние клиента.

1. Использование JSON — данные чаще всего передаются в формате

JSON.

Архитектура обеспечивает масштабируемость и минимальную задержку,

сохраняя возможность расширения функционала.

**2.4 Паттерны проектирования в разработке системы**

1. Builder:

- использовал паттерн Builder при создании тестовых пользователей в

классе DataInitializer для инициализации базы данных при пустом состоянии. Это позволило удобно задавать значения полей, избегая перегрузки конструкторов.

1. Repository:

- интерфейсы UserRepository, MessageRepository и др. реализуют паттерн Repository, предоставляя абстракцию над базой данных. Через них осуществляется автоматическое создание таблиц, выполнение запросов и взаимодействие с сущностями через Spring Data JPA.

1. Service Layer:

- паттерн Service Layer реализован в классах UserService,

MemoryUserServiceImpl, UserServiceImpl, которые обрабатывают бизнес-логику. Сервисы вызываются из контроллеров и работают с DTO-классами, инкапсулируя логику вне слоя представления.

1. Observer:

- реализовал паттерн Observer для уведомления пользователей о новых

сообщениях и чатах. Например, при отправке сообщения вызывается метод, уведомляющий пользователей через SseController.

1. Transactional:

- аннотацией @Transactional пометил методы в сервисе

UserServiceImpl, где происходит работа с несколькими сущностями. Это обеспечивает целостность данных при выполнении операций, особенно при массовом обновлении или удалении.

1. DTO (Data Transfer Object):

- используются DTO-классы UserRepository, MessageRepository,

InMemoryUserDAO для передачи данных между контроллерами и сервисами. DTO отделяют сущности JPA от внешнего слоя, предотвращая утечку бизнес-логики и избыточной информации.

1. Singleton:

- все сервисы и репозитории (UserService, MessageRepository и др.)

внедряются как Singleton-бины по умолчанию через аннотации @Service, @Repository. Это обеспечивает единый экземпляр на всё приложение и централизованный доступ к функциональности.

1. Strategy:

- интерфейс UserService реализован двумя классами:

UserServiceImpl и MemoryUserServiceImpl. В контроллере выбирается реализация в зависимости от контекста, что соответствует паттерну Strategy — поведение уведомления зависит от стратегии.

1. Command

- каждый REST-запрос, обрабатываемый контроллерами

(UserController, SseController), представляет собой реализацию паттерна Command — инкапсуляция действия (например, EntranceByEmail, Registration) как команды, вызываемой с параметрами.

1. MVC (Model-View-Controller)

- использую структуру MVC: @RestController (SseController) обрабатывают входящие HTTP-запросы, передают данные в сервисный слой, который работает с моделями и репозиториями.

**3 Разработка приложения**

В данном разделе описывается процесс разработки серверной (бэкенд) и клиентской (фронтенд) частей приложения. В качестве стека технологий были выбраны: Java + Spring Boot для бэкенда и Thymeleaf в качестве шаблонизатора для отображения данных на веб-страницах. Такой подход позволяет реализовать серверный рендеринг и полностью контролировать поток данных между клиентом и сервером.

* 1. **Реализация бэкенда**

Бэкенд приложения был реализован с использованием фреймворка Spring Boot, который обеспечивает быструю настройку, автоконфигурацию и поддержку REST API. Основные задачи бэкенда:

- управление бизнес-логикой;

- взаимодействие с базой данных через JPA/Hibernate;

- обработка HTTP-запросов и маршрутизация;

- возврат данных или представлений (в случае Thymeleaf).

Ключевые компоненты:

- Сущности (Entities) — классы, отражающие структуру таблиц в базе данных.

- Репозитории (Repositories) — интерфейсы, предоставляющие доступ к данным.

- Сервисы (Services) — слой бизнес-логики, обрабатывающий запросы и координирующий работу с данными.

- Контроллеры (Controllers) — принимают запросы от пользователя, вызывают соответствующие методы сервисов и передают данные на фронтенд.

Пример контроллера:

@Controller  
@RequestMapping("/api/v1")

@AllArgsConstructor

public class UserController {

@Autowired  
 private SseController sseController;  
 private final UserService userService;

// выводит список пользователей и обрабатывает пагинацию пользователей, а так же добавляю пользователей в беседу  
 @GetMapping("/users")  
 public String FindAllUsers()

// обеспечивает окно для авторизации пользователей  
 @GetMapping("/entrance")  
 public String SimpleReturnShape()

// авторизация пользователей  
 @PostMapping("/entranceByEmail")  
 public String EntranceByEmail()

// предоставляет окно, который ведёт к методу Registration для регистрации новых пользователей  
 @GetMapping("/registrationForm")  
 public String RegistrationForm()

// обеспечивает добавление новых пользователей  
 @PostMapping("/registration")  
 public String Registration()

// Обеспечивает поиск пользователей по email, пагинацию обрабатывая в FindAllUsers по session  
 @PostMapping("/findByEmail")  
 public String FindUserByEmail()

// А это маппер для вывода сообщений  
 @GetMapping("/users/{id}")  
 public String ShapeFormForId()

// Я уже начинаю теряться, это маппер для принятия и ввода сообщений  
 @PostMapping("/users/{id}/message")  
 public String InputMessages()

// выводит список с пользователями, с которыми есть переписка, а также может выводить список бесед  
 @GetMapping("/historyOfMessages")  
 public String historyOfMessages()

// метод, который может обновлять данные пользователя  
 @PostMapping("/updateUser")  
 public String updateUser()

// метод, который активирует окно с допольнительным подтверждением о удалении аккаунта пользователя  
 @GetMapping("deleteUser")  
 public String deleteUserByEmail()// само окно с дополнительным поддтверждение о удалении аккаунта пользователя (тут удаляется аккаунт)

@GetMapping("deleteUser/True")  
 public String deleteUserByEmailIfTrue()  
 // ссылка для выхода из аккаунта пользователя  
 @GetMapping("exitFromAccaunt")  
 public String exitFromAccaunt()

// метод, который получает данные для создания беседы  
 PostMapping("/historyOfMessages/createOfConversionPost")  
 public String createOfConversionAction()

// Метод для отправки сообщения в беседу  
 @PostMapping("/conversation/{id}/sendMessage")  
 public String conversionSendMessage()

// сообщения беседы, будут дублированы на примере ShapeFormForId +++  
 @GetMapping("/conversation/{id}")  
 public String conversion()

// Метод, который будет демонстрировать список собеседников и представит возможность добавления новых пользователей  
 @GetMapping("/conversation/{id}/ListOfUsers")  
 public String getListOfUsers()

// метод для добавления пользователей  
 @GetMapping("/conversation/{id}/addUsers")  
 public String getAddUser()

}

Контроллеры — это классы, которые обрабатывают входящие HTTP-запросы, вызывают бизнес-логику и возвращают результат пользователю.

Контроллер в архитектуре Spring Boot служит интерфейсом между внешними клиентами (например, фронтендом, мобильным приложением, другими микросервисами) и внутренними слоями приложения.

В контексте сервисной архитектуры контроллер:

- Обрабатывает входящие HTTP-запросы (REST API).

- Делегирует обработку данных бизнес-логике, которая реализуется в сервисном слое.

- Возвращает клиенту структурированный HTTP-ответ (обычно в формате JSON).

- Не содержит бизнес-логики — только маршрутизация, валидация, и вызов сервисов.

Пример сервиса:

public interface UserService {

// метод предоставляет список пользователей добавленных в беседу  
 List<User> getListUsersForAddConversation();

// метод предоставляет список всех зарегистрированных пользователей  
 List<User> FindAllUsers();

// метод ищет пользователя по уникальному идентификатору  
 User FindUserById(long id);

// метод ищет сообщения по уникальному идентификатору  
 Message FindMessageById(long id);

// метод предоставляет список всех сообщений  
 List<Message> getAllOfMessage();

// метод создающий новое сообщение  
 Message AddMessage(Message message);

// метод удаляющий сообщение по уникальному идентификатору  
 void DeleteMessage(long id);

// метод сохраняющий нового пользователя  
 User SaveUser(User user);

// метод ищет пользователя по электронной почте и паролю  
 User FindUserByEmailAndPassword(String email, String password);

// метод, которые позволяет удалить пользователя  
 void DeleteUser(String email);

// метод, которые ищет беседу по названию  
 Conversation FindConversationById(String name);

// метод, который создаёт новую беседу  
 Conversation setNewConversation(Conversation newConversation);

// метод, который предоставляет весь список бесед  
 List<Conversation> FindAllConversations();

// метод, который удаляет беседу по уникальному идентификатору  
 void deleteConversationById(long id);

// метод, который обновляет данные беседы  
 Conversation UpdateConversation(Conversation newConversation);

// метод, который создаёт новый список с беседой и состоящих в ней пользователей  
 MidConversation setNewMidConversation(MidConversation newConversation);

// метод, который выдаёт все списки бесед и состоящих в них пользователей  
 List<MidConversation> FindAllMidConversations();  
}

@Service — это специализированная аннотация Spring, обозначающая, что класс реализует бизнес-логику приложения. Такие классы автоматически регистрируются как Spring-бины и участвуют в внедрении зависимостей (@Autowired, constructor injection и т.д.).

Назначение слоя Service:

- Инкапсуляция бизнес-логики. Контроллер ничего не должен "знать" о том, как обрабатываются данные — он просто вызывает методы сервиса.

- Связующее звено между контроллером и репозиторием.

- Обработка условий, проверок, трансформаций данных, вызовов сторонних API и т. д.

- Поддержка транзакций (через @Transactional, если нужно).

- Повторное использование логики между разными частями системы.

Пример репозитория:

public interface UserRepository extends JpaRepository<User, Long> {

// МЕТОД, КОТОРЫЙ ИЩЕТ ПОЛЬЗОВАТЕЛЯ ПО ЭЛЕКТРОННОЙ ПОЧТЕ В БАЗЕ ДАННЫХ  
 User findByEmail(String email);

// МЕТОД, КОТОРЫЙ ИЩЕТ ПОЛЬЗОВАТЕЛЯ ПО ЭЛЕКТРОННОЙ ПОЧТИ И ПАРОЛЮ В БАЗЕ ДАННЫХ  
 User findByEmailAndPassword(String email, String password);

// МЕТОД ВЫДАЮЩИЙ СПИСОК ПОЛЬЗОВАТЕЛЕЙ С ПРИМЕНЕНИЕМ ПАГИНАЦИИ  
 Page<User> findAll(Pageable pageable);  
}

Репозитории в Spring Boot (с использованием Spring Data JPA) отвечают за прямое взаимодействие с базой данных. Они абстрагируют низкоуровневую работу с SQL и позволяют работать с БД через интерфейсы и методы.

Задачи репозитория:

- выполнение CRUD-операций (Create, Read, Update, Delete).

- Поиск данных по заданным параметрам (например, через findByName, findByStatusAndType и т.д.).

- Работа с базой через JPA/Hibernate.

- Не содержат бизнес-логики — только доступ к данным.

**3.2 Разработка фронтенда**

Фронтенд приложения реализован с использованием серверного шаблонизатора Thymeleaf, который интегрируется с Spring MVC и позволяет напрямую связывать данные с HTML-шаблонами. Thymeleaf поддерживает динамическую генерацию HTML-страниц на основе переданных с сервера данных.

Особенности использования Thymeleaf:

- Поддержка Spring EL (Expression Language) для вставки переменных: ${user.name}

- Динамические конструкции: th:each, th:if, th:href, th:action и др.

- Прямая интеграция с моделью Spring (Model, ModelMap)

Интеграция с бэкендом организована через REST API: создание сессии,

отправка действий и получение результатов. Обработка ошибок выводит

пользователю уведомления, сохраняя стабильность интерфейса.  
Преимущества выбора Thymeleaf:

- Простота интеграции с Spring;

- Возможность использовать шаблоны как статически, так и динамически;

- Ясный и читаемый синтаксис;

- Удобная отладка HTML-прототипов прямо в браузере.

Интеграция с бэкендом организована через REST API: создание сессии,

отправка действий и получение результатов. Обработка ошибок выводит

пользователю уведомления, сохраняя стабильность интерфейса.

1. **Тестирование**

Тестирование программного обеспечения (ПО) является неотъемлемой частью процесса разработки и обеспечивает высокое качество конечного продукта. Тестирование выполняется для проверки соответствия ПО требованиям, выявления ошибок, дефектов и недоработок, а также для проверки функциональности, производительности и безопасности.

Роль тестирования ПО заключается в том, чтобы помочь разработчикам и менеджерам проекта убедиться в том, что ПО работает должным образом и соответствует ожиданиям заказчика. Тестирование также помогает выявить ошибки и дефекты, которые могут привести к сбоям в работе ПО или даже к возможным угрозам безопасности.

**4.1 Unit-тесты**

Юнит-тесты (unit tests) — это тесты, которые проверяют работу отдельных единиц кода (например, функций, методов или классов) в изоляции от остальной системы. Основная цель — проверить, что каждая часть программы работает корректно.

Пример кода для Unit-тестов:

@ExtendWith(MockitoExtension.class)  
class SimpleChatNewJava17ApplicationTests {

// мок объект для эмуляции работы сервиса  
 @Mock  
 private UserService userService;

// мок объект для эмуляции работы модели шаблонизатора  
 @Mock  
 private Model model;

// мок объект для эмуляции работы сессий   
 @Mock  
 private HttpSession session;

// мок объект для эмуляции работы главного контроллера  
 @InjectMocks  
 private UserController userController;

// метод тестирует авторизацию  
 @Test  
 public void testEntranceByEmail\_SuccessfulLogin()

// метод тестирует регистрацию с неверным проверочным паролем  
 @Test  
 void registration\_PasswordMismatch\_ShouldReturnRegistrationView()

// метод тестирует полную регистрацию  
 @Test  
 void registration\_NewUser\_ShouldSaveAndReturnEntrance()

// Тестируем проверку на сходие email  
 @Test  
 void registration\_EmailAlreadyRegistered\_ShouldReturnRegistrationWithError()

// протестировали поиск  
 @Test  
 void testFindUserByEmail\_NoUsersFound\_ShouldAddErrorAndAttributesToModel()

// метод, тестирующий отправку новых сообщений

@Test  
 void testInputMessages()

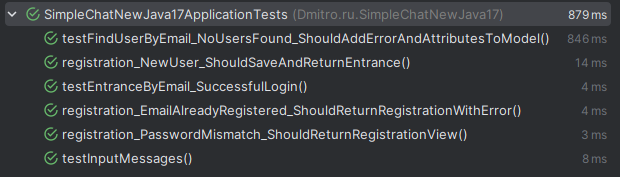


Рисунок 1 – Успешная работа тестов.

**4.2 Проверка функциональности веб-приложения**

Это проверка будет заключаться в тестировании функционала самого сайта, с результатом после каждого действия. Итоги этого теста будут отражены в самом конце в таблице. Проверка программы представлена в рисунках 7-25.

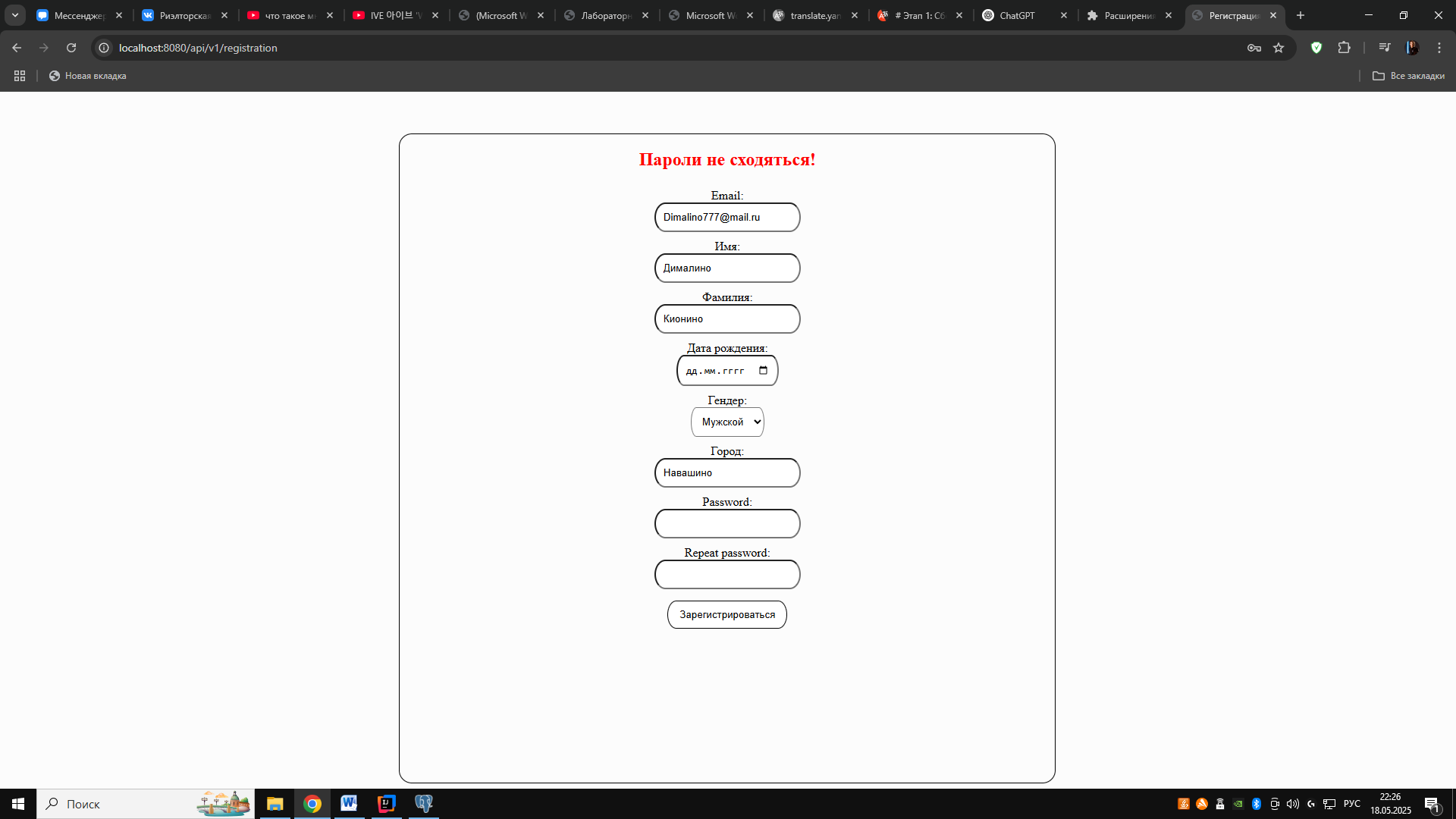


Рисунок 2 – Успешная защита при регистрации от неправильного ввода проверочного пароля.

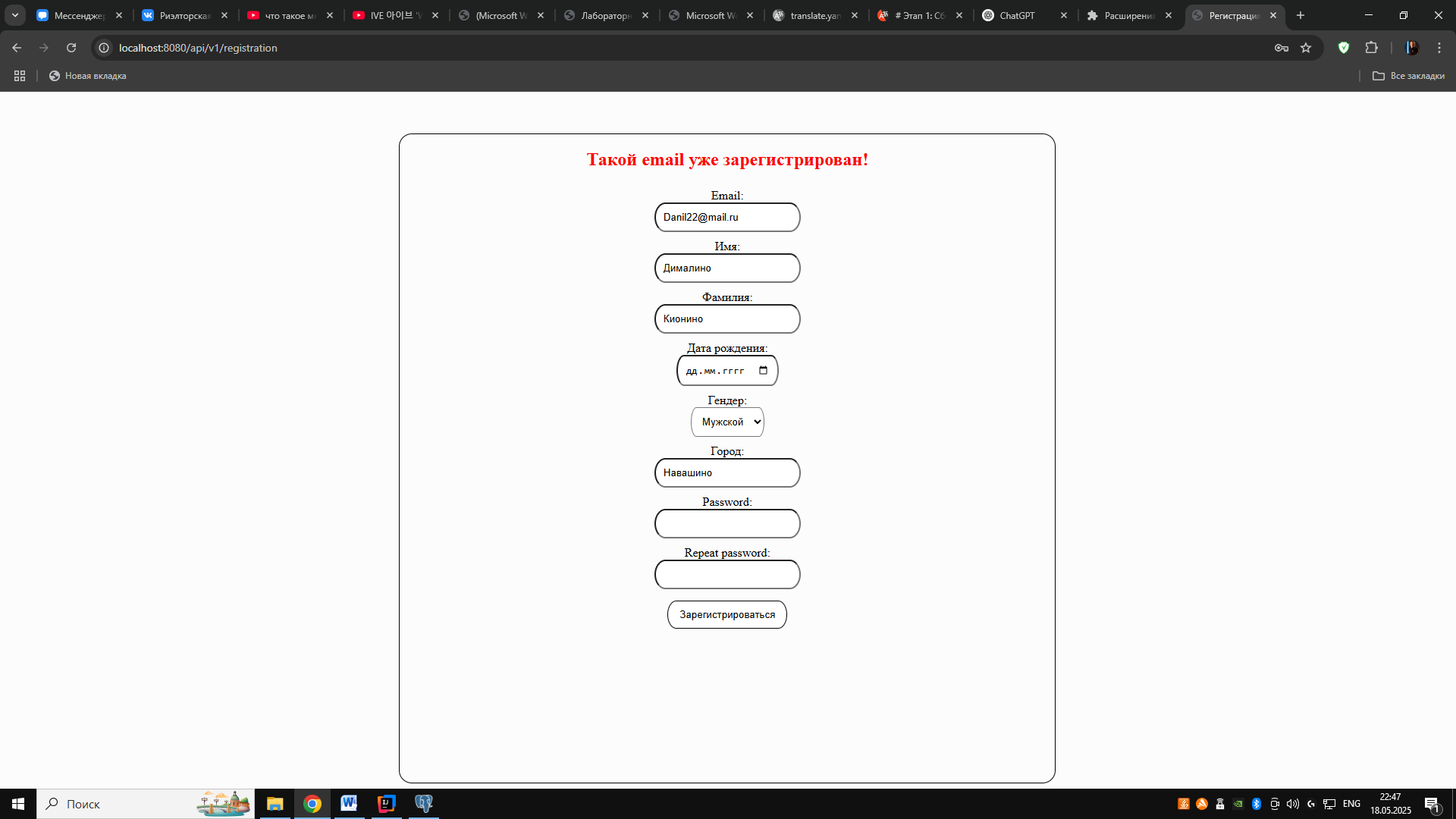


Рисунок 3 – Успешная защита при регистрации от ввода существующего email.

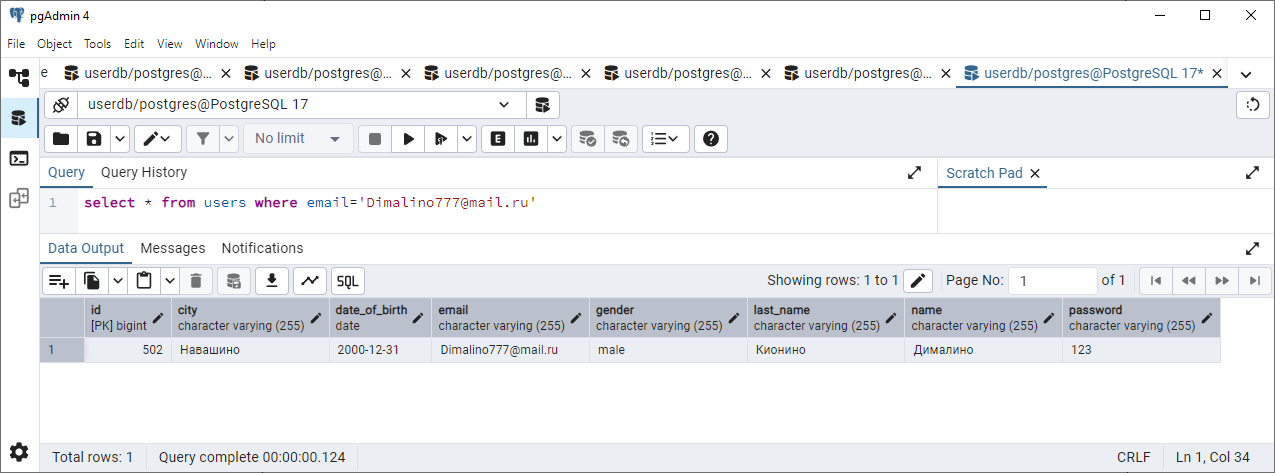


Рисунок 4 – Успешный ввод данных в базу данных при правильной регистрации.

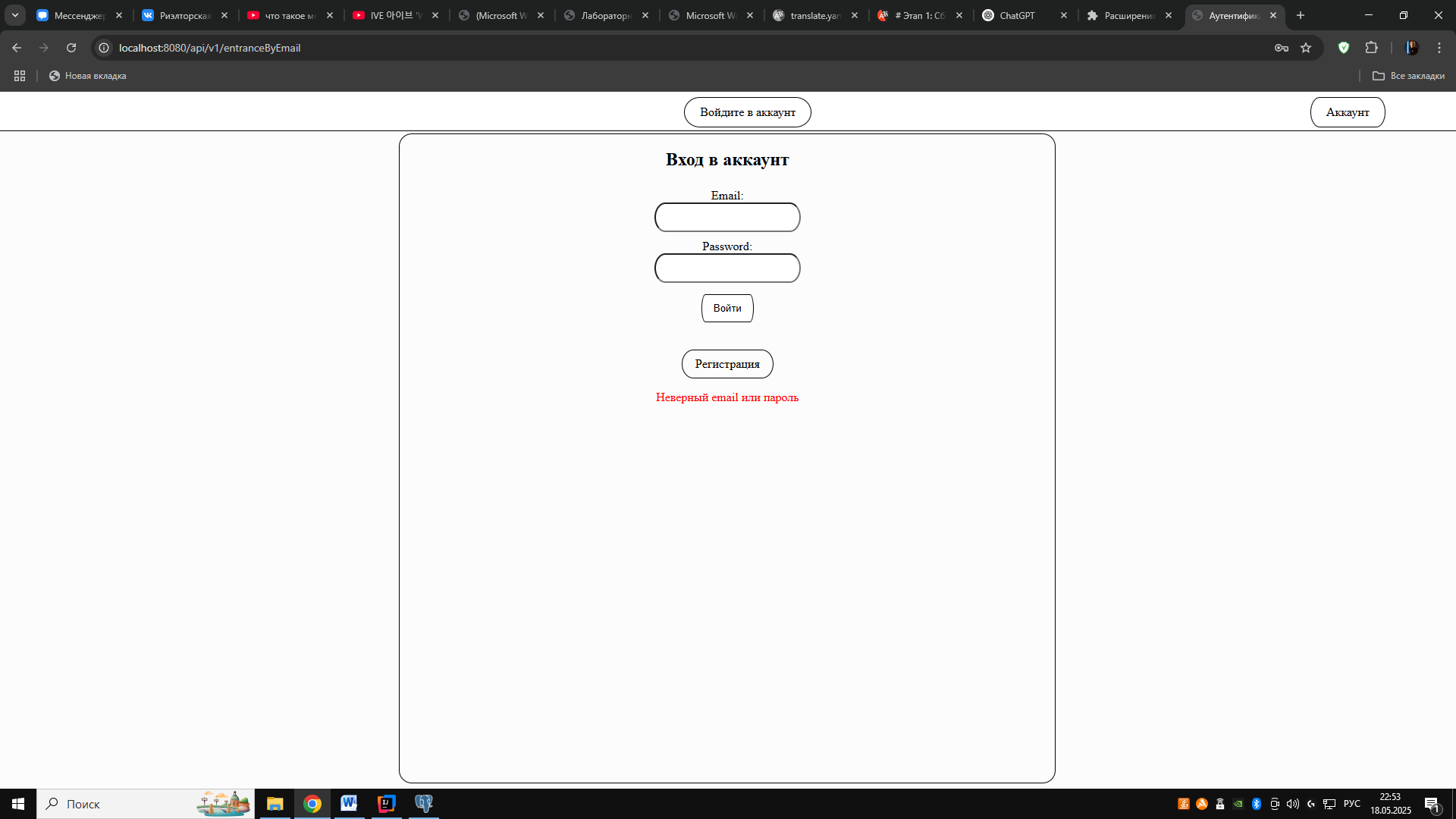


Рисунок 5 – Успешная защита при аутентификации пользователя при неправильном вводе пароля или email.

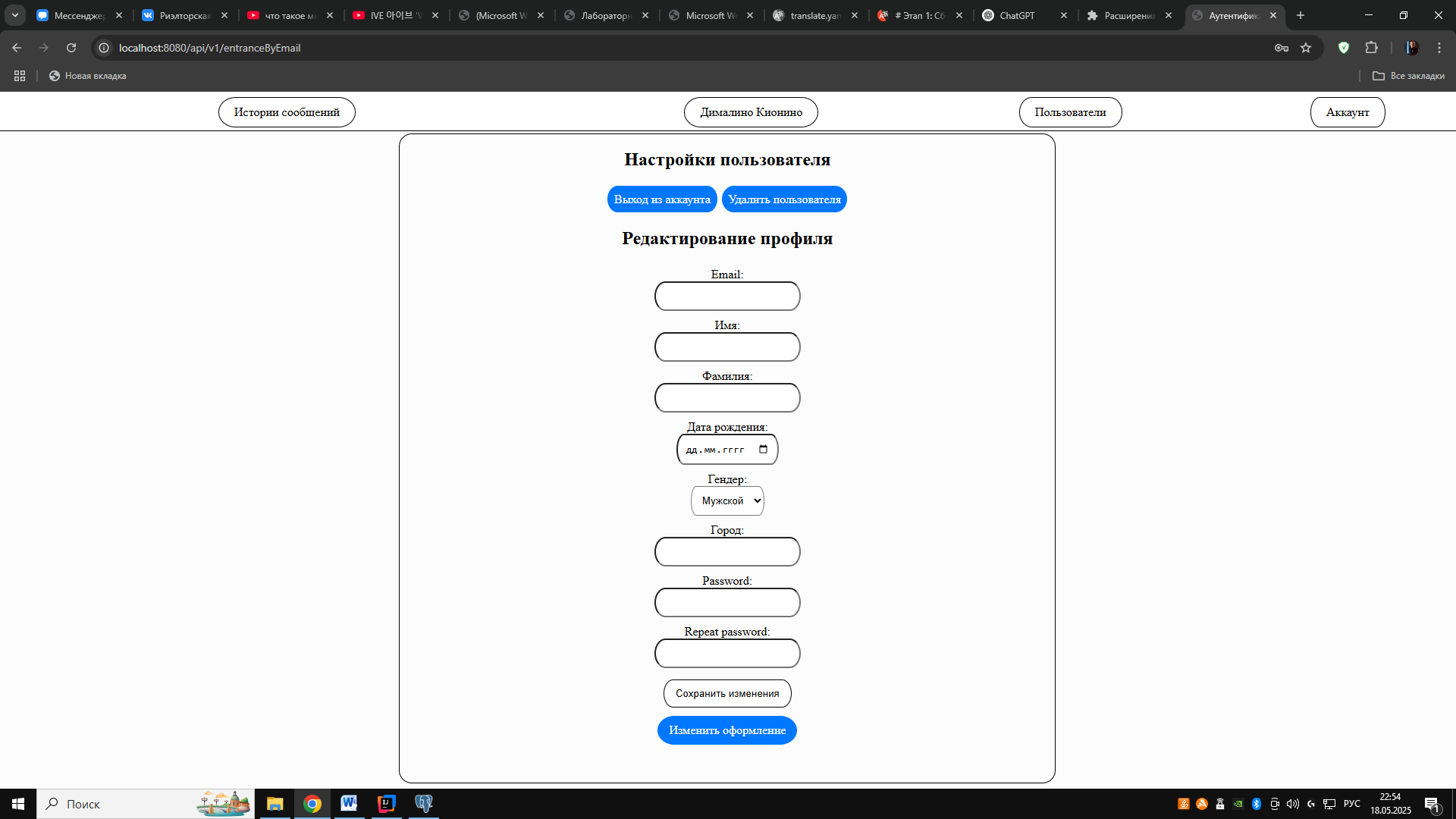


Рисунок 6 – Успешная аутентификация пользователя.

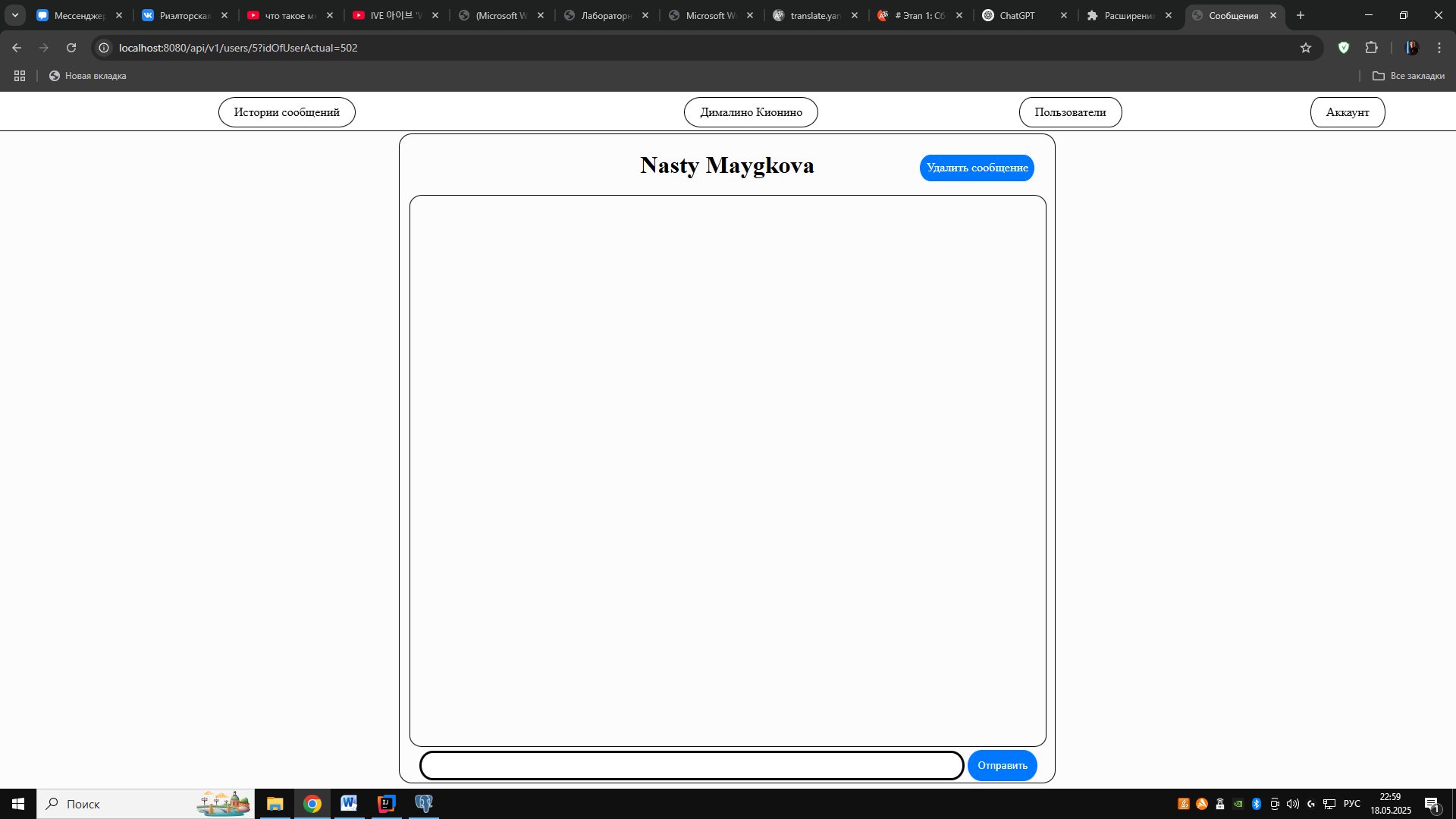


Рисунок 7 – Успешное открытие сообщений с пользователем.

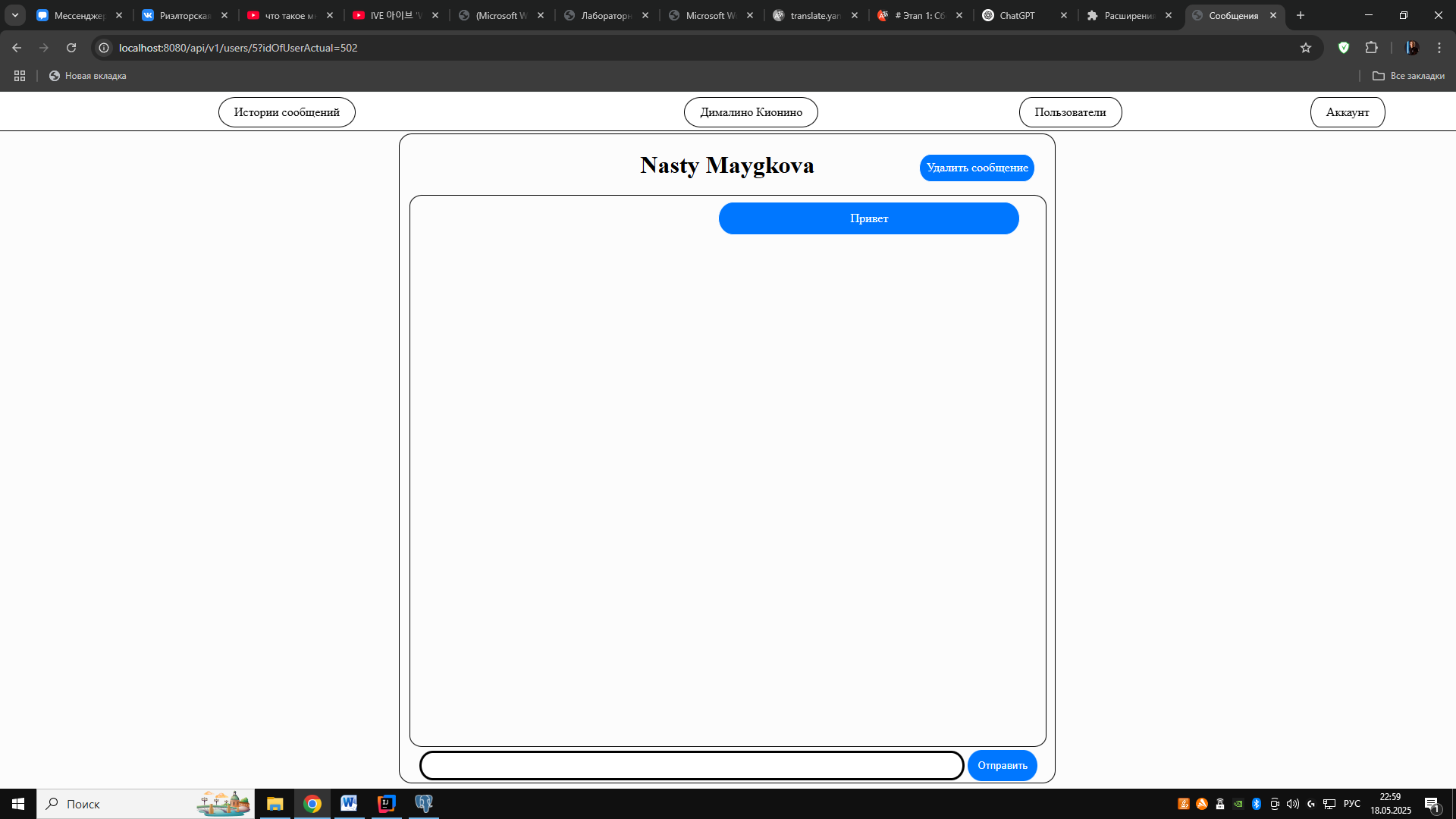


Рисунок 8 – Успешно отправил сообщений пользователю.

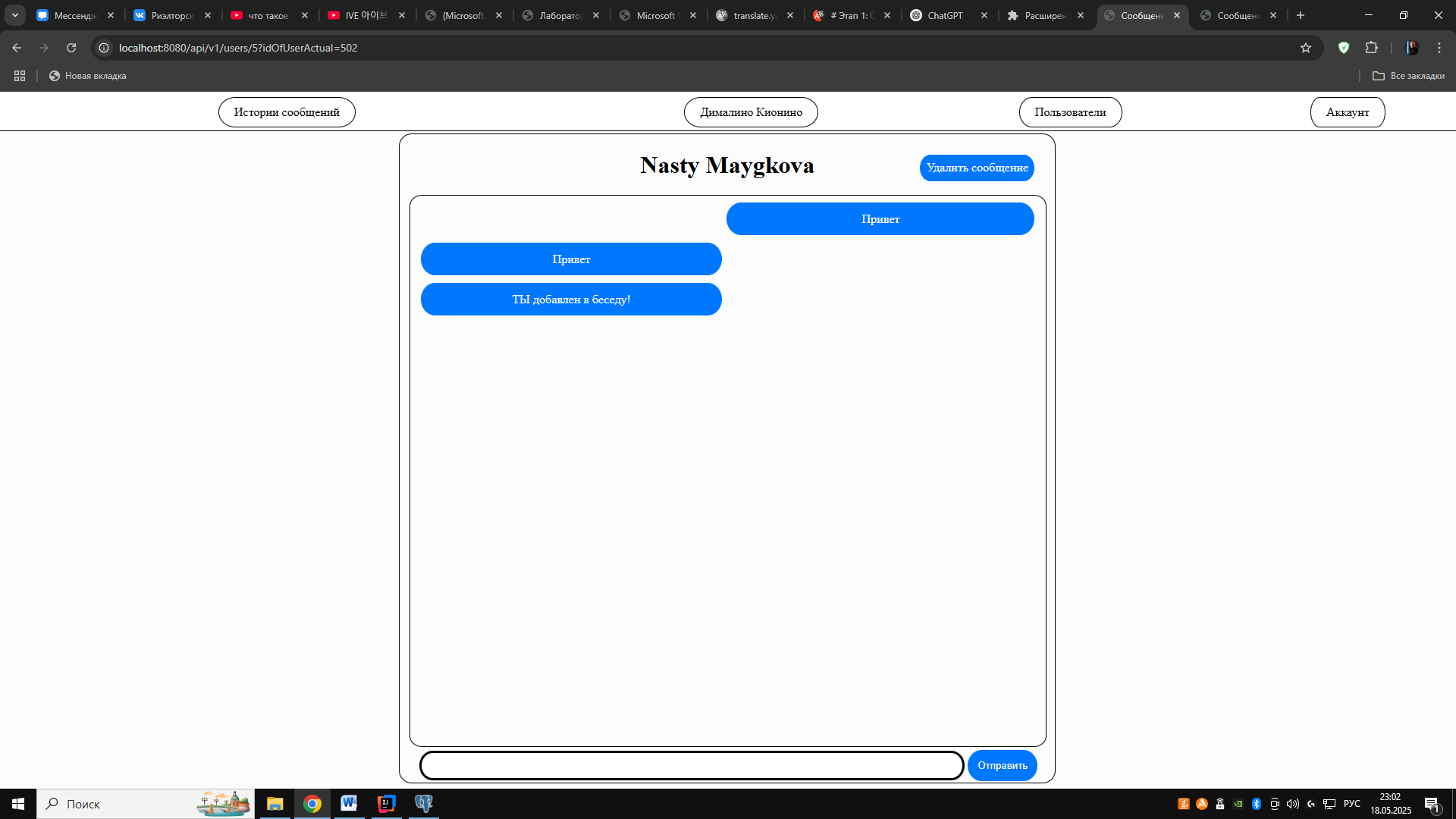


Рисунок 9 – Успешно получил сообщение от пользователя.

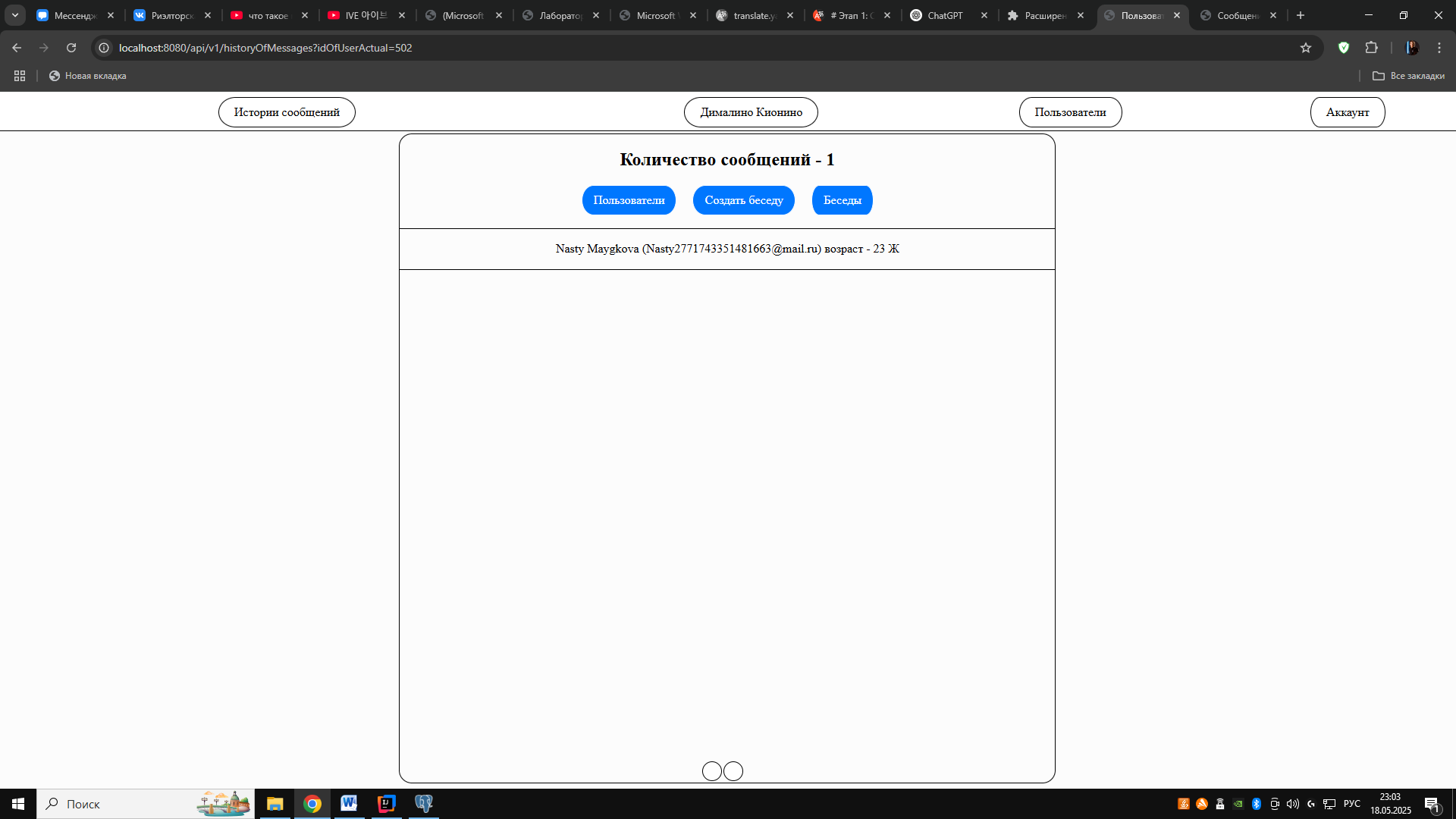


Рисунок 10 – Успешно открыл историю переписок с пользователями.

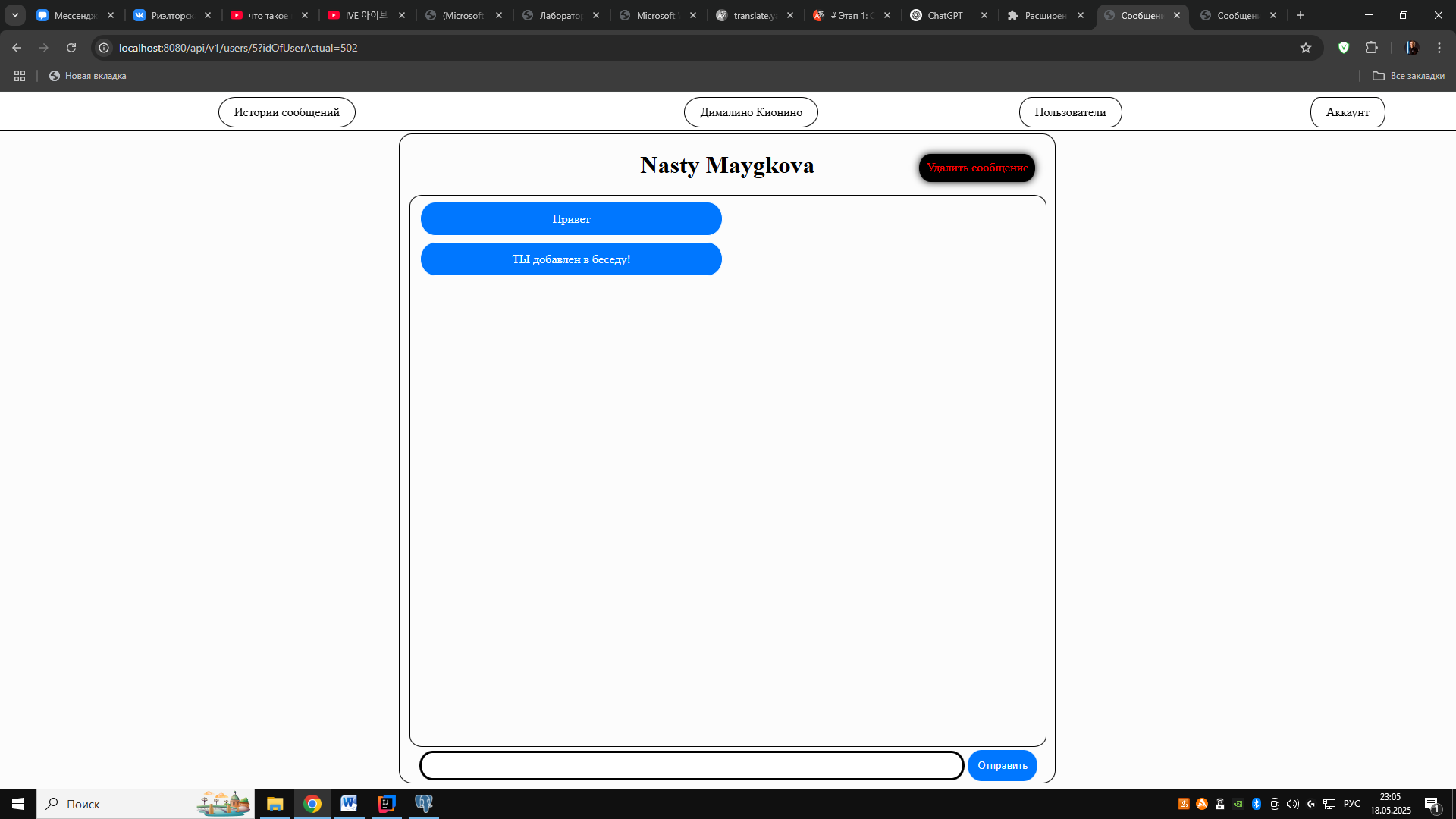


Рисунок 11 – Успешно удалил своё первое сообщение.

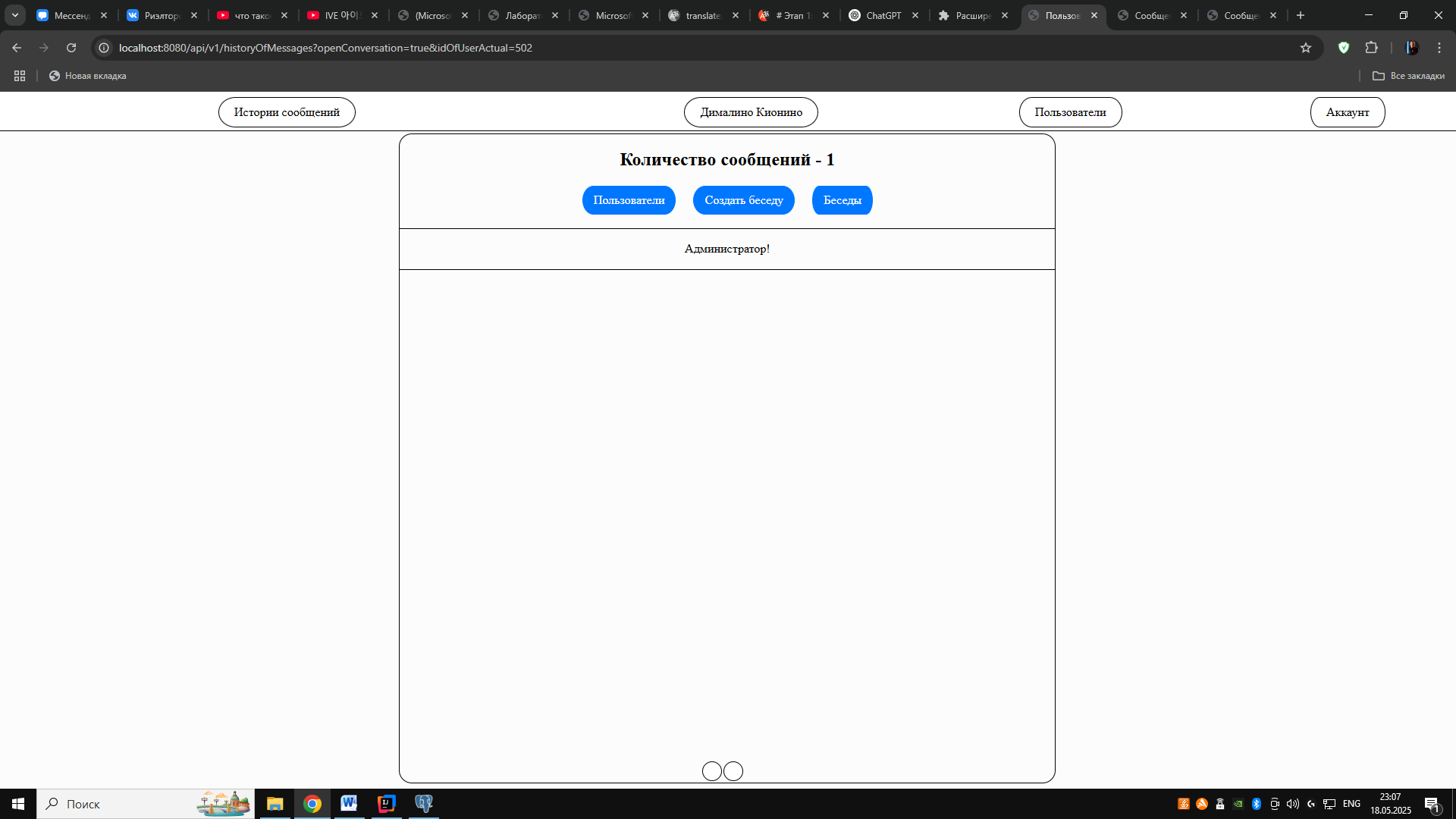


Рисунок 12 – Успешно отрыл историю бесед, в который состоит мой пользователь.

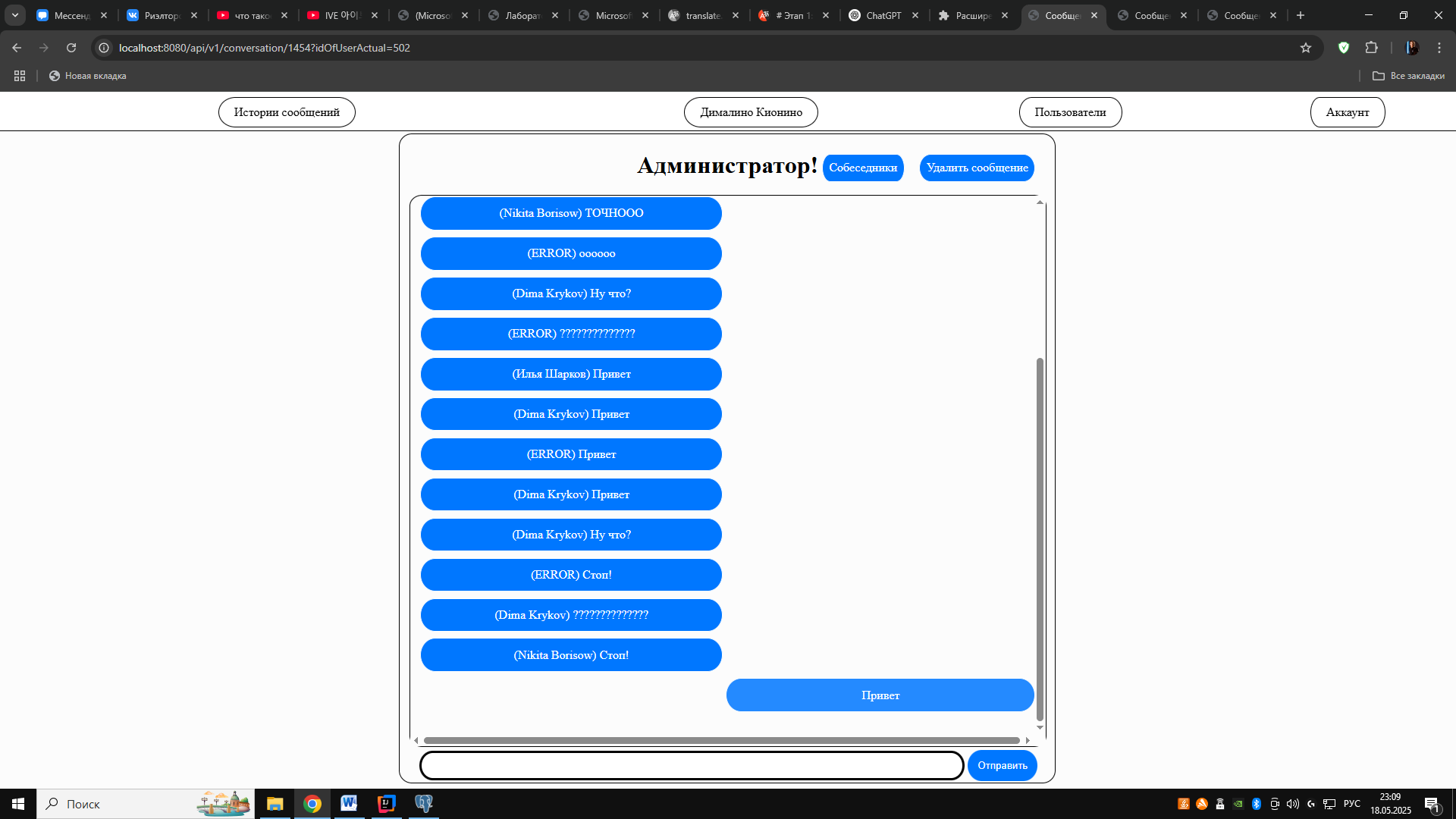


Рисунок 13 – Успешно отправил сообщение в беседу.

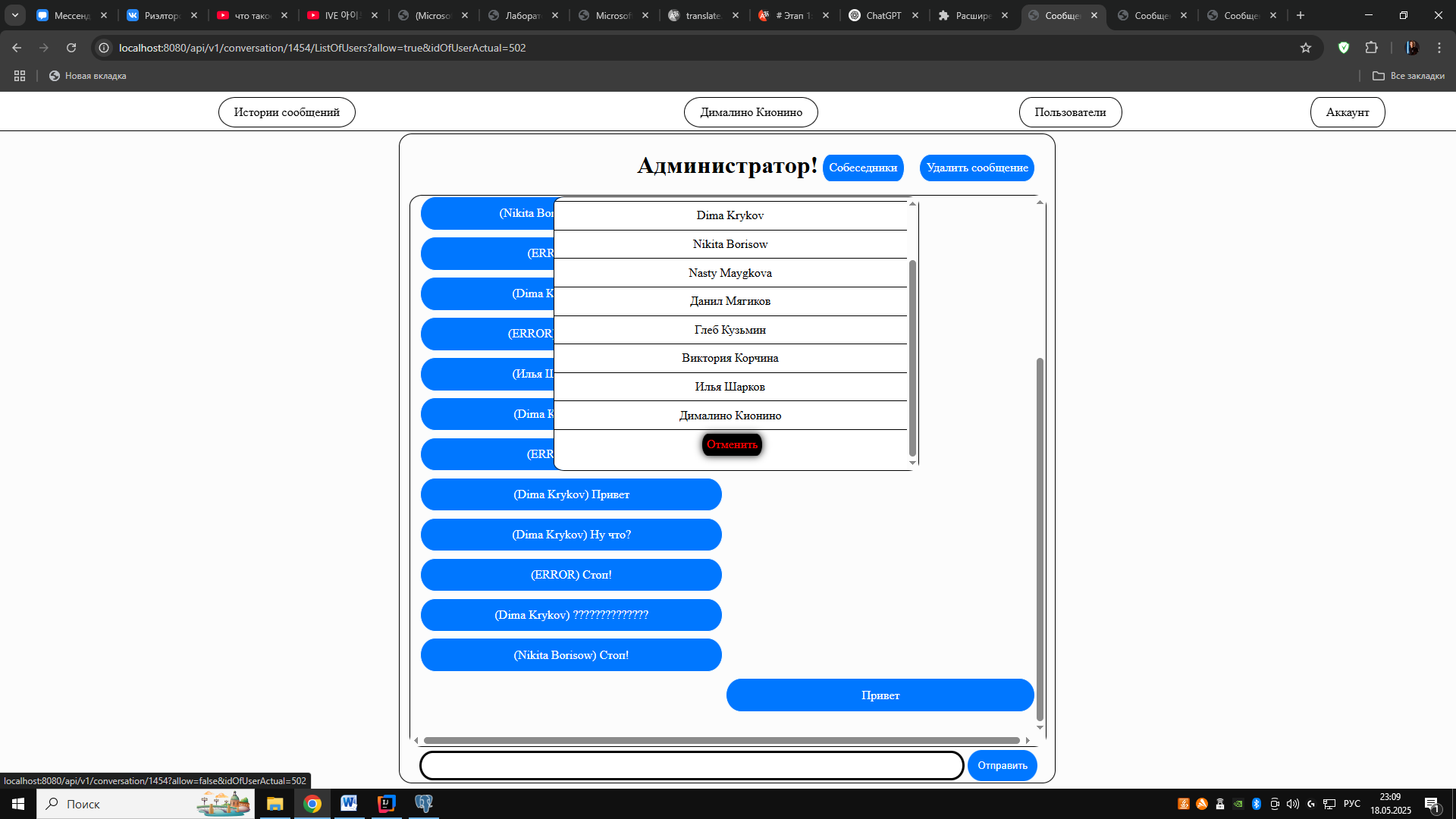


Рисунок 14 – Успешно проверил список пользователей состоящих в беседе, а так же ограничение на добавление новых пользователей.

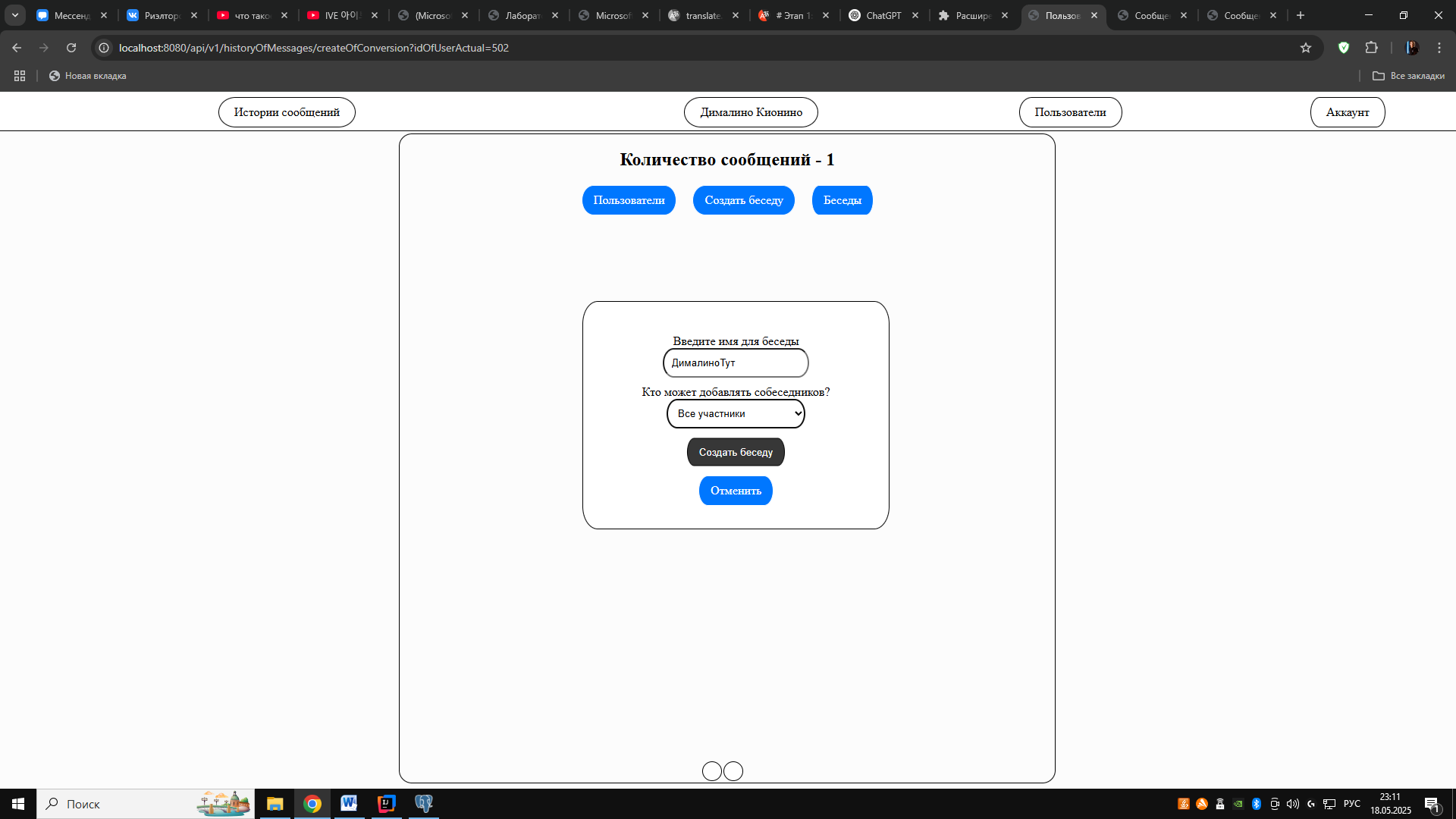


Рисунок 15 – Успешно открыл окно с созданием беседы.

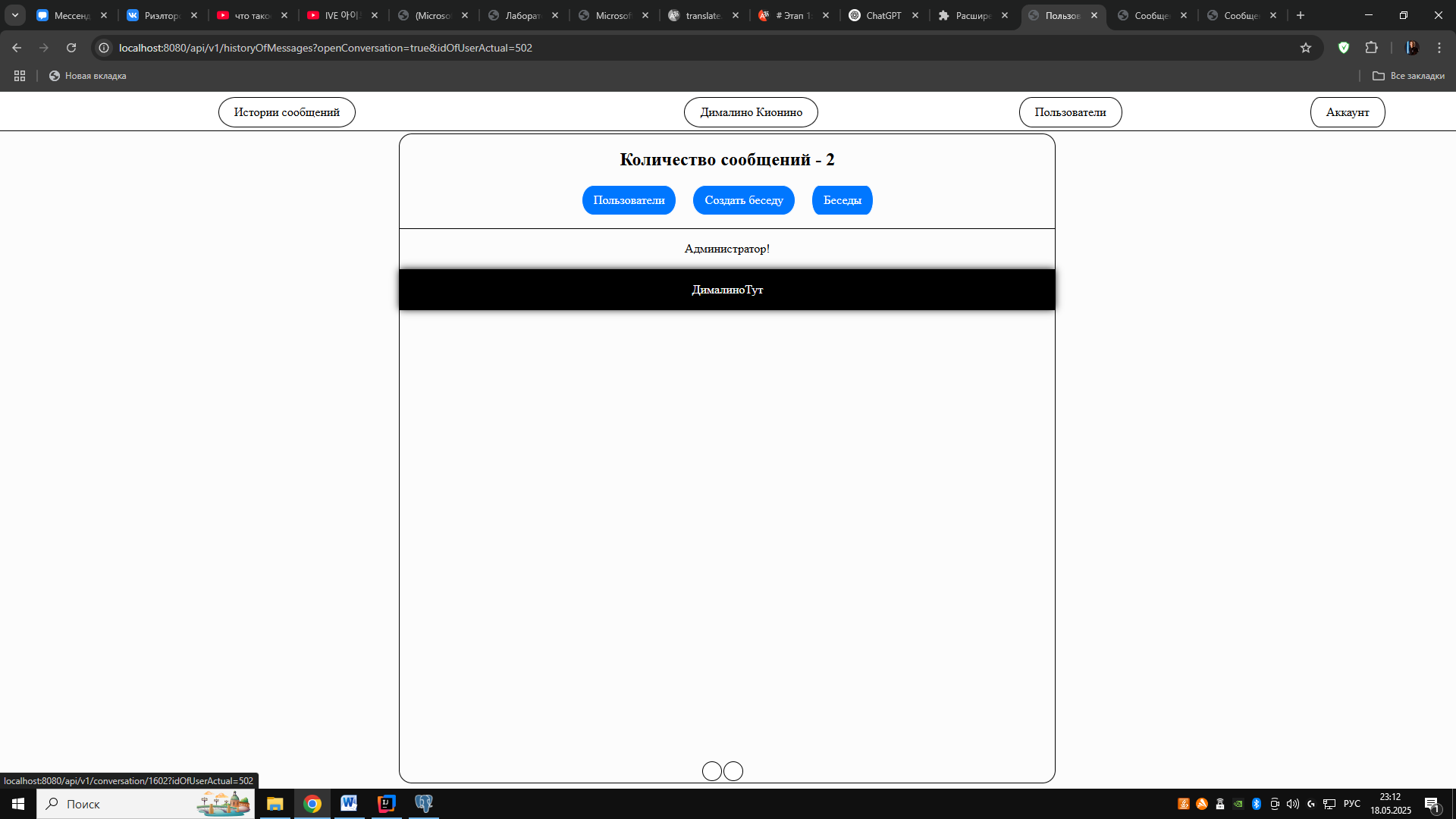


Рисунок 16 – Успешно создал новую беседу.

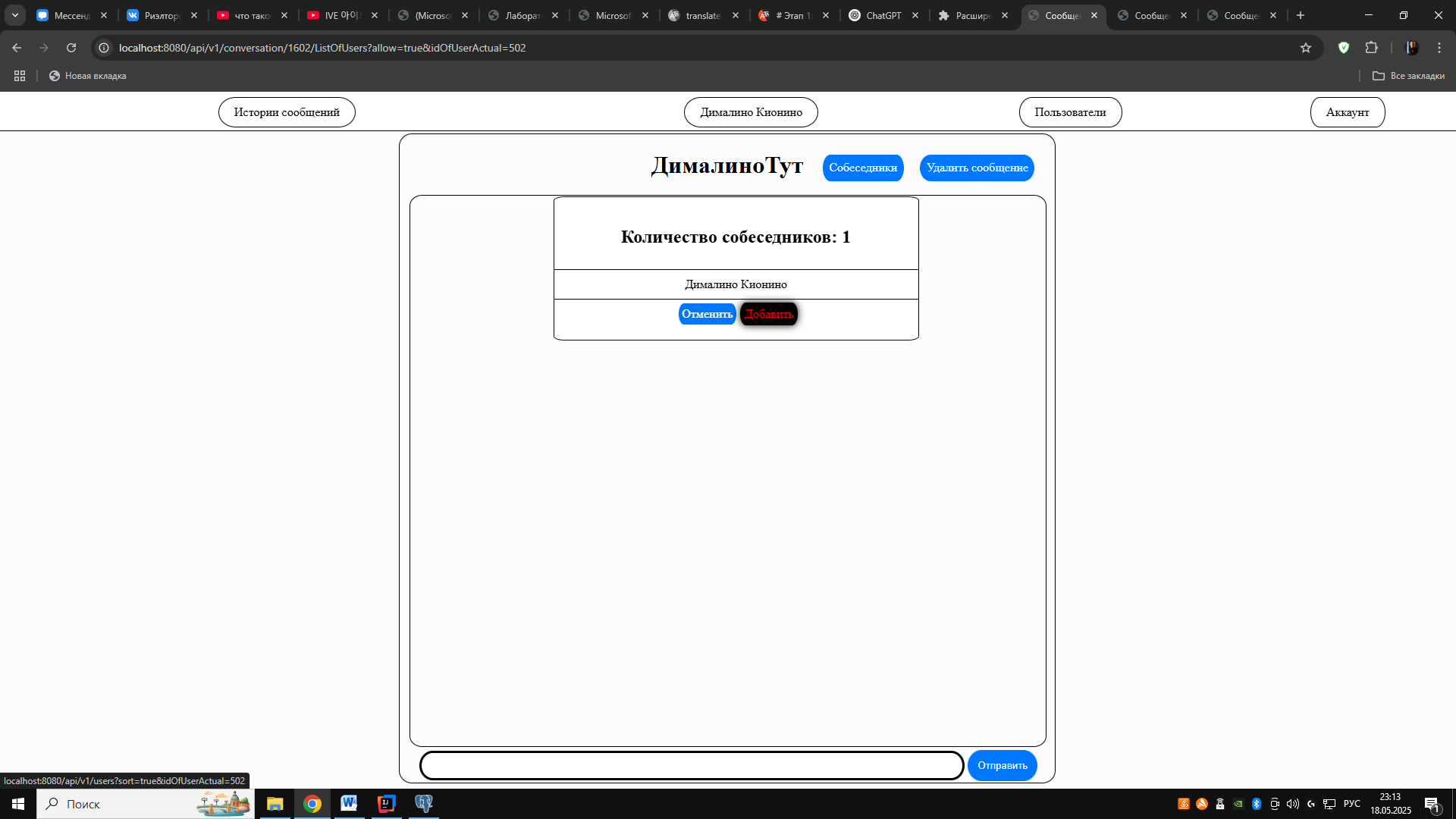


Рисунок 17 – Успешно открыл список пользователей находящих в беседе и проверил возможность для добавления новых пользователей в беседу.

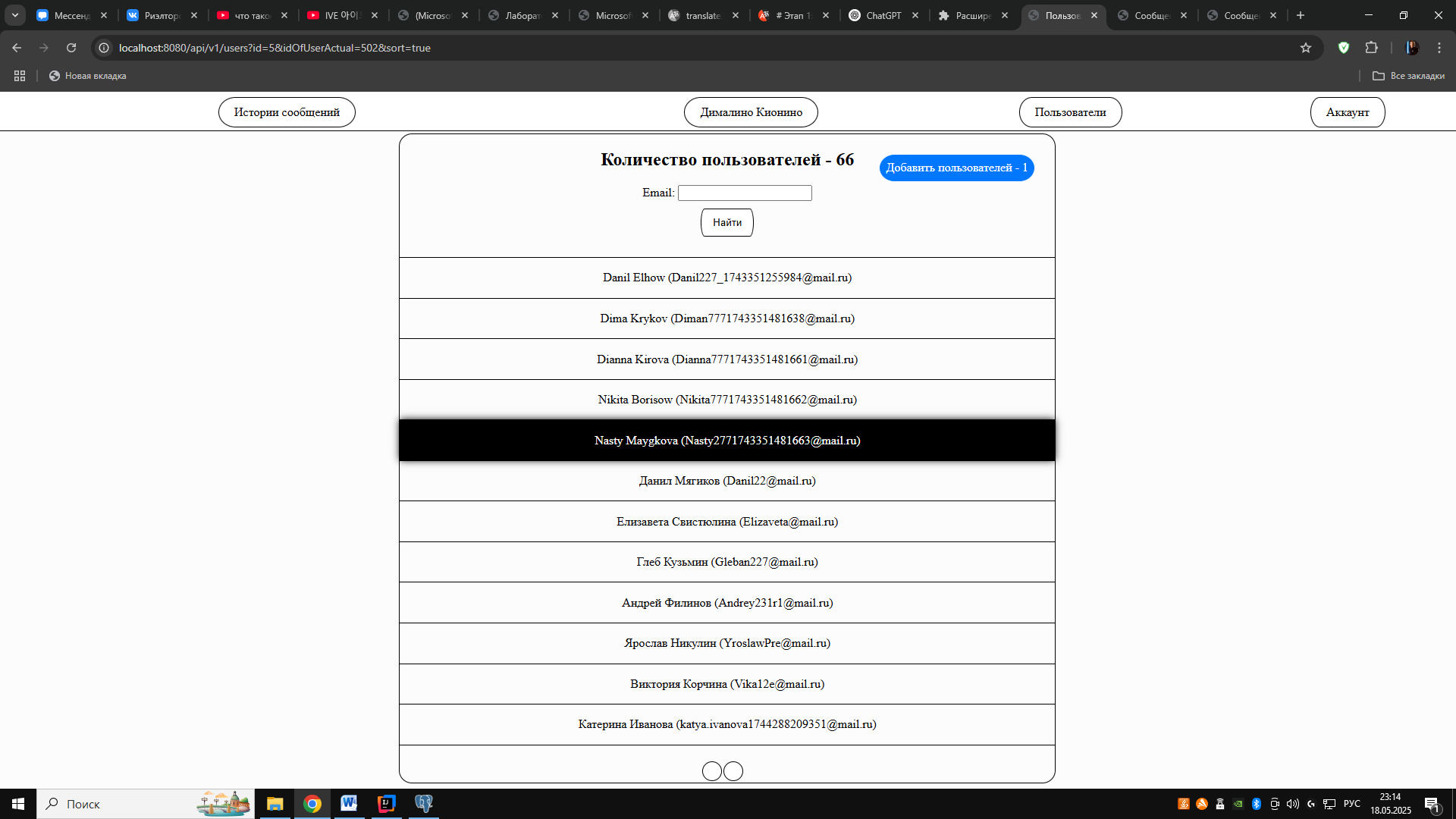


Рисунок 18 – Успешно кликнул на пользователя для добавления его в свою беседу.

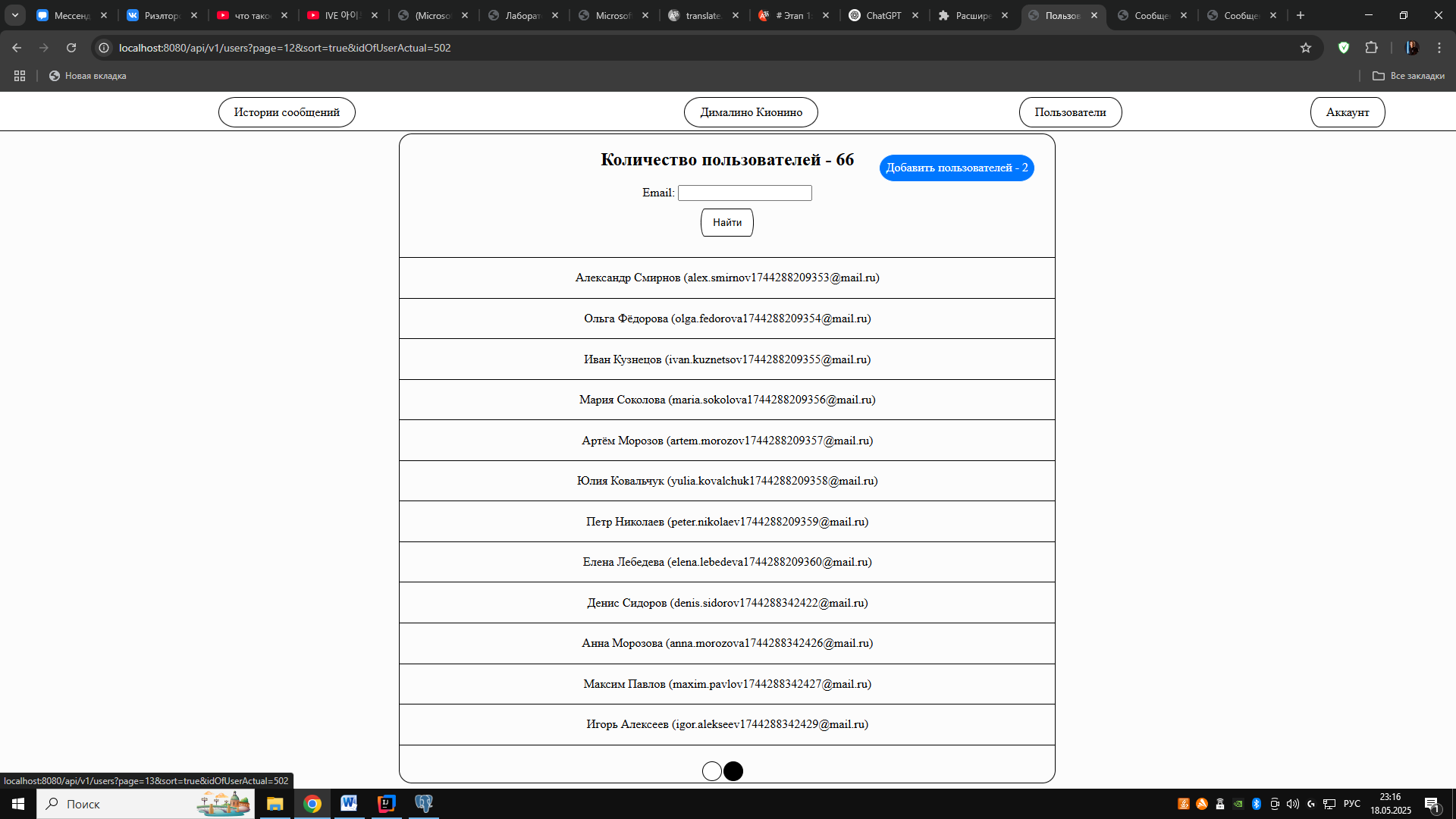


Рисунок 19 – Успешно протестировал пагинацию для списка пользователей.

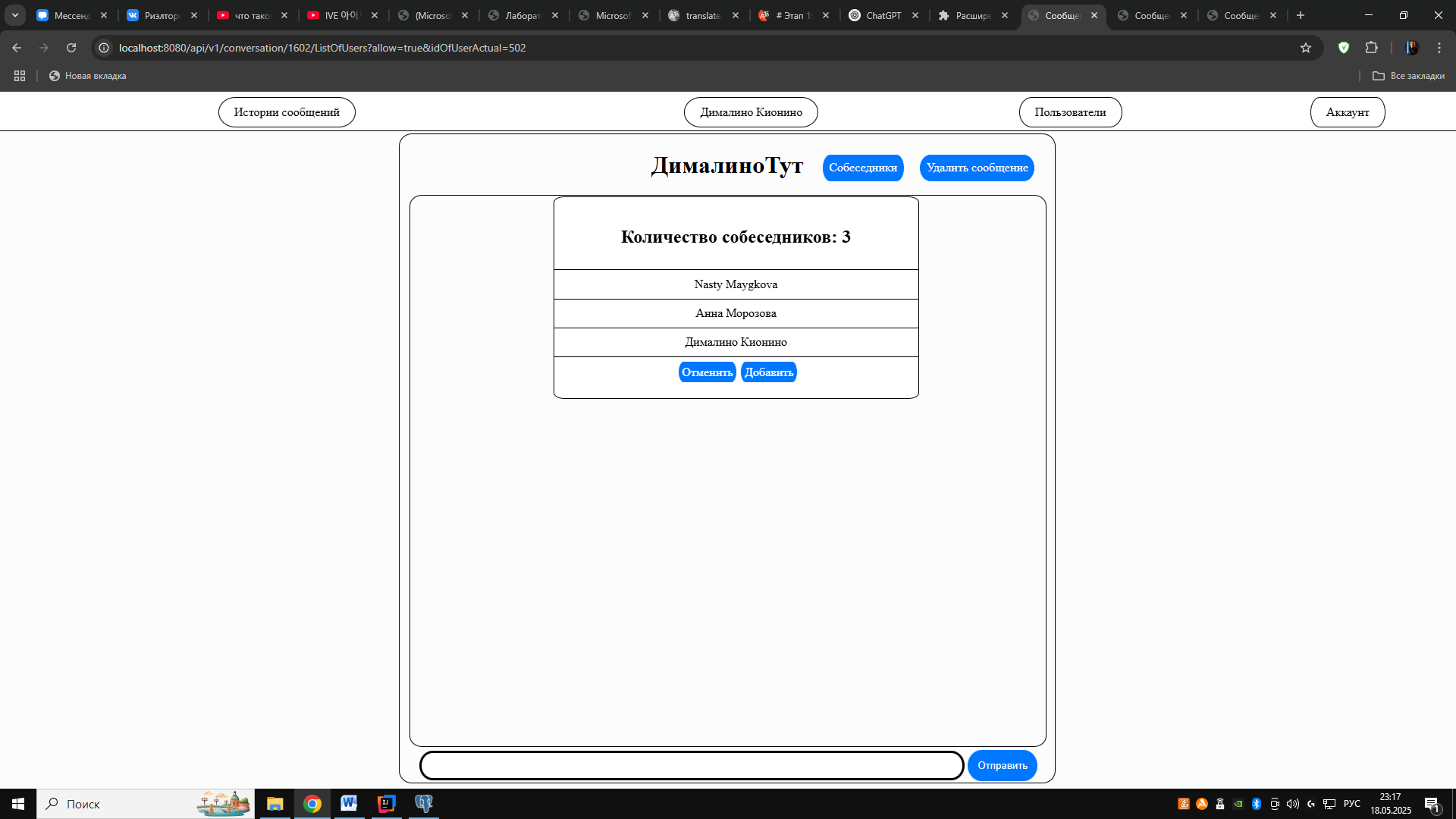


Рисунок 20 – Успешно проверил наличие новых пользователей в беседе.

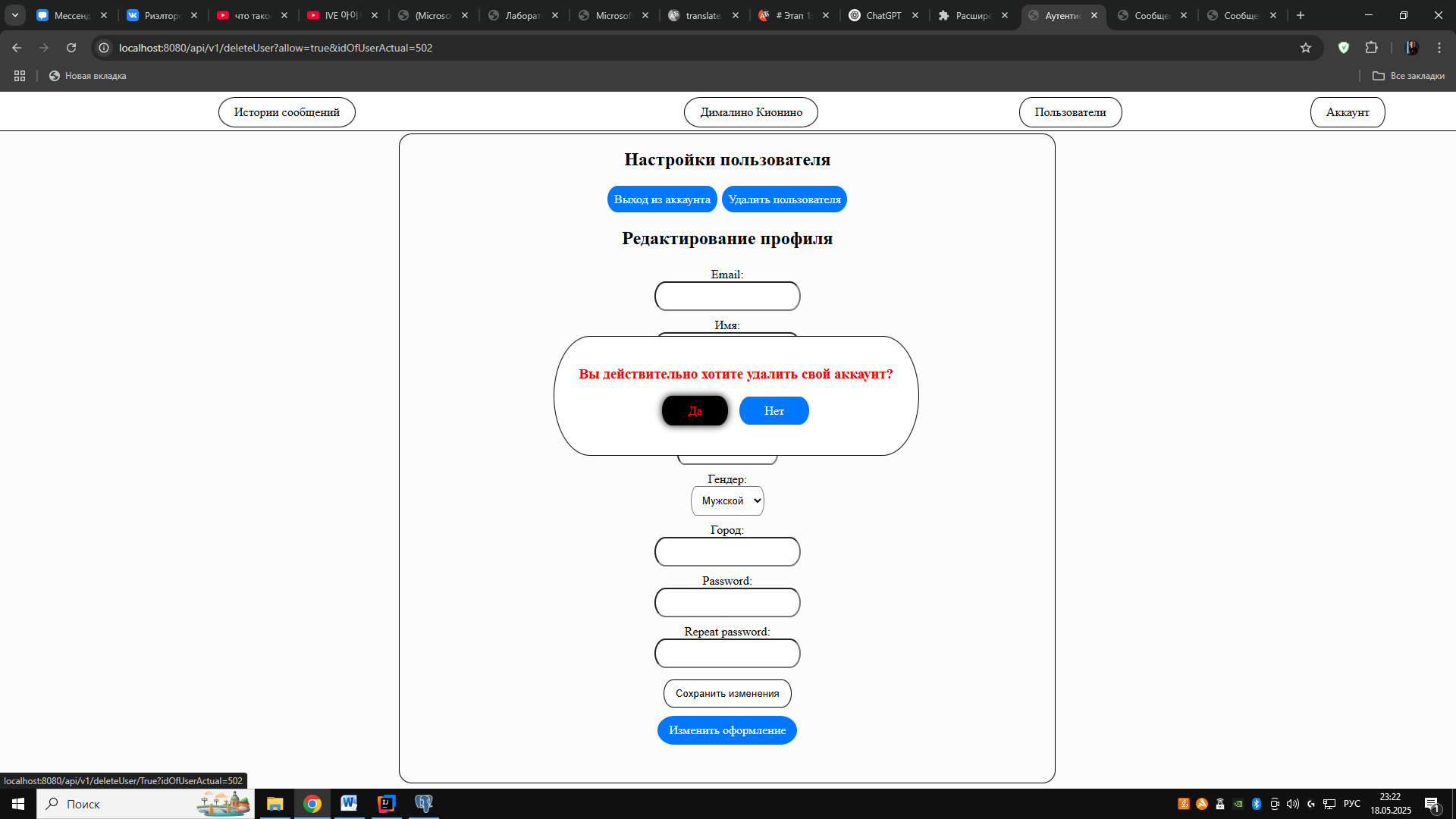


Рисунок 21 – Успешно открыл окно с дополнительным подтверждением для удаления аккаунта.

Таблица 1 — Тестирование программы.

|  |  |  |
| --- | --- | --- |
| **№** | **Функция** | **Результат** |
| 1 | Защита при регистрации от неправильного ввода проверочного пароля. | **+** |
| 2 | Защита при регистрации от ввода существующего email. | + |
| 3 | Ввод данных в базу данных при правильной регистрации. | + |
| 4 | Защита при аутентификации пользователя при неправильном вводе пароля или email. | + |
| 5 | Аутентификация пользователя. | + |
| 6 | Открытие сообщений с пользователем. | + |
| 7 | Отправка сообщений пользователю. | + |
| 8 | Принятие сообщений от пользователя. | + |
| 9 | Открытие истории переписок с пользователями. | + |
| 10 | Удаление сообщений пользователя | + |
| 11 | Открытие списка бесед, в которых состоит пользователь. | + |
| 12 | Отправка сообщение в беседу. | + |
| 13 | Проверка списка пользователей состоящих в беседе, а так же ограничение на добавление новых пользователей. | + |
| 14 | Активация окна с созданием беседы. | + |
| 15 | Создание новой беседы. | + |
| 1617 | Проверка снятие ограничений на добавление пользователей. | + |
| 17 | Клик на пользователя для добавления его в свою беседу. | + |
| 18 | Пагинация для списка пользователей. | + |
| 19 | Активация окна с дополнительным подтверждением для удаления аккаунта. | + |
| 20 | Удаление данных аккаунта из базы данных | + |

По результатам тестирования выявлено, что программа не имеет логических ошибок.

Данная программа тестировалась на компьютере следующей конфигурации:

1. Процессор: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz;
2. ОЗУ: 16.00 Гб;
3. ОС: Windows 10 64-разрядная.

**ЗАКЛЮЧЕНИЕ**

В данной курсовой работе были углублены знания, полученные в ходе лекционных и практических занятий по управлению данными

В ходе выполнения курсовой работы были приобретены навыки работы с библиотеками spring boot на языке java. Было создано веб-приложение «Социальная сеть», которое предоставляет возможность пользователям регистрироваться, общаться по сети, искать новых собеседников и создавать беседы, а так же для удобства была добавлена возможность для удаления своих неудачных сообщений, во избежание неприятных казусов, которые могли бы опорочить впечатление от использования данного приложения.

В рамках работы последовательно были выполнены следующие этапы: проанализирована предметная область, разработаны варианты использования программы, определены основные таблицы и их поля. Затем была создана база данных, а также разработан графический интерфейс приложения. Завершающим этапом стало функциональное тестирование, подтвердившее корректность работы программы.

В дальнейшем возможно усовершенствование программы как по функциональным, так и оптимизационным параметрам путем добавления новых алгоритмов, новых полезных функций, реализации задуманных ранее, но в последствии отклонённых функций и оптимизации уже существующего кода.

Кроме того, в процессе разработки была приобретена ценная практика работы с архитектурными паттернами и принципами построения масштабируемых приложений, что позволит в будущем создавать более надежные и эффективные системы.

Особое внимание уделялось вопросам безопасности данных и удобству пользовательского взаимодействия, что способствует формированию положительного опыта использования приложения.

**СПИСОК ЛИТЕРАТУРЫ**

1. Блох, Джошуа. Эффективное программирование на Java. Москва: ДМК пресс 2020 456 с. [Электронный ресурс]. - <https://ftp.zhirov.kz/books/IT/Java/Java.%20%D0%AD%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%28%D0%94%D0%B6%D0%BE%D1%88%D1%83%D0%B0%20%D0%91%D0%BB%D0%BE%D1%85%29.pdf>.
2. Еккаль, Брюс. Философия Java. Санкт-Петербург: Питер, 2019, 1162 с. [Электронный ресурс]. - <https://www.rulit.me/books/filosofiya-java-read-180907-1.html>.
3. Хоретман, Кей С. Биг Java: ранние объекты Москва – Питер, 2020 1200 с. [Электронный ресурс]. - <https://www.rulit.me/author/horstmann-kej-s/java-biblioteka-professionala-tom-1-osnovy-11-e-izd-download-677514.html>.
4. Котляр, Виктор. Spring Framework. Разработка приложений на Java. Москва: ДМК Пресс, 2021. 560 с. [Электронный ресурс]. - <https://hsse.spbstu.ru/userfiles/files/Sovremennie-tehnologii-(v-sbore).pdf>.
5. Фаулер, Мартин. Паттерны проектирования. Санкт-Петербург: Питер, 2018. 320 с. [Электронный ресурс]. - <https://martinfowler.com/books/eaa.html>.
6. Ричардсон, Кевин. Spring Boot в действии. Санкт-Петербург: Питер, 2020. 384 с. [Электронный ресурс]. - <https://market.yandex.ru/product--spring-v-deistvii-shestoe-izdanie/1769944382?sku=101817961324&uniqueId=182454729&do-waremd5=yoowWmsDzkccAZtC4Dn1qw>.
7. Spring Framework Official Documentation [Электронный ресурс]. - <https://spring.io/projects/spring-boot>.
8. PostgreSQL Documentation [Электронный ресурс]. -<https://www.postgresql.org/docs/>.
9. Oracle. Java Documentation [Электронный ресурс]. -<https://docs.oracle.com/en/java/>.
10. Thymeleaf Official Documentation [Электронный ресурс]. -<https://www.thymeleaf.org/documentation.html>.