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"

– Разработана веб-приложение с интуитивным управлением;

– Проведена отладка и тестирование приложения для различных не запланированных 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, указано 12 таблиц и 24 иллюстраций.

– Приведены примеры работы программы.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc105757734)

[1 Анализ технического задания 5](#_Toc105757735)

[1.1 Общее описание системы 5](#_Toc105757741)

[1.2 Технические требования 6](#_Toc105757736)

[1.3 Функциональные требования](#_Toc105757737) 6

[1.4 Нефункциональные требования 7](#_Toc105757737)

[1.5 Архитектура приложения 7](#_Toc105757737)

[1.6 Выбор технологий и инструментов 7](#_Toc105757737)

[1.7 Процесс разработки](#_Toc105757737) 8

[1.8 Ожидаемые результаты](#_Toc105757737) 8

[2 Проектирование системы](#_Toc105757739) 8

2.1 [Model (Модель)](#_Toc105757739) 8

[2.2 Controller (Контроллер)](#_Toc105757739) 9

[2.3 Service (Сервис) 11](#_Toc105757739)

[2.4 Репозиторий (Repository) 1](#_Toc105757739)3

[2.5 Краткое сравнение ролей слоёв 1](#_Toc105757739)4

[2.6 Итоговая архитектура Spring Boot приложения: 1](#_Toc105757739)4

[3 Проектирование системы 1](#_Toc105757740)5

3.1 Структура проекта……………………………………………………………………………15

3.2 База данных……. ……....………………………………………………………………...16

3.2.1 Список таблиц с полями и типами данных..………………………………...16

3.2.2 ORM-модели………………………………...…………………………………17

3.3 Технологический стек……........…………………………………………..……………...20

3.4 Паттерны проектирования в разработке системы..……………………………….20

3.5 Unit-тесты…………………………………………...……………………………….22

[4 Тестирование](#_Toc105757742) 26

[5 Руководство по программному продукту 38](#_Toc105757743)

[5.1 Руководство программиста 38](#_Toc105757743)

[5.2 Руководство администратора 38](#_Toc105757743)

[5.3 Руководство пользователя 39](#_Toc105757743)

[Заключение 40](#_Toc105757744)

[Список литературы](#_Toc105757744) 41

**Введение**

В последние десятилетия 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.

**1 Анализ технического задания**

По заданию необходимо создать веб-приложение «Социальная сеть», которая будет копировать часть основного функционала из других социальных сетей. В качестве языка программирования будет 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
* Поиск пользователей по email

Модели данных и их связи:

* Users: (long) id, name, (String) last\_name, (LocalDate) date\_of\_birth, (String) gender, (String) city, (String) email, (String) password, (int) age.
* Messages: (long) id, (String) message, (long) firstID (Users), (long) secondID (Users).
* Conversations: (long) id, (String) message, (String) nameOfConversation, (long) IDOwner (Users), (boolean) adminIsOwner.
* Midconversations: (long) id, (long) IdOfUser (Users), (long) IdOfConversation (Conversations).

**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 Выбор технологий и 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.

**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.

**2. Проектирование системы**

В разработанном веб-приложении используется клиент-серверная архитектура, в которой клиент (пользователь через браузер) взаимодействует с сервером (бэкендом на Spring Boot) по протоколу HTTP.

Архитектура приложения построена на основе шаблона проектирования MVC (Model–View–Controller), который способствует разделению обязанностей между компонентами системы:

**2.1 Model (Модель)**

Модель представляет бизнес-логику и данные приложения. В моей системе это классы-сущности, которые отображаются на таблицы в базе данных. Например:

* User — информация о пользователе,
* Messages — переписка между двумя пользователями,
* Conversations — беседа со множеством пользователей.
* Midconversations — информация о нахождении пользователей в беседах

Каждая сущность аннотирована как @Entity и управляется через JPA/Hibernate.

**2.2 Controller (Контроллер)**

Аннотация: @Controller

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

Описание:

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 на главный контроллер по обработке HTTP-запросы:

Таблица 1 – код главного контролера без реализации методов и поясняющими комментариями (часть 1).

|  |
| --- |
| @Controller @RequestMapping("/api/v1")  @AllArgsConstructor  public class UserController { |

Продолжение на следующей странице…

Таблица 1.2 – код главного контролера без реализации методов и поясняющими комментариями (часть 2).

|  |
| --- |
| @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() |

Продолжение на следующей странице…

Таблица 1.3 – код главного контролера без реализации методов и поясняющими комментариями (часть 3).

|  |
| --- |
| // 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() |

**2.3 Service (Сервис)**

Аннотация: @Service

@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 на интерфейс для сервиса:

Таблица 2 – код интерфейса для работы с сервисом.

|  |
| --- |
| public interface UserService {  List<User> getListUsersForAddConversation();  List<User> FindAllUsers();  User FindUserById(long id);  User FindUserByEmail(String email);  Page<User> FindAllUsers(int page, int size);  Message FindMessageById(long id);  List<Message> getAllOfMessage();  Message AddMessage(Message message);  Message UpdateMessage(long id, String newMessageContent);  void DeleteMessage(long id);  User SaveUser(User user);  User FindUserByEmailAndPassword(String email, String password);  User UpdateUser(User user);  void DeleteUser(String email);  User getInMemoryUser();  void setInMemoryUser(User user);  Conversation FindConversationById(String name);  Conversation setNewConversation(Conversation newConversation);  List<Conversation> FindAllConversations();  void deleteConversationById(long id);  Conversation UpdateConversation(Conversation newConversation);  MidConversation FindMidConversationById(String nameOfConversation, long idOfUser);  MidConversation setNewMidConversation(MidConversation newConversation);  List<MidConversation> FindAllMidConversations();  void deleteMidConversationById(long id); } |

**2.4 Репозиторий (Repository)**

Описание:

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.

Пример на репозиторий для обработке SQL запросов для класса User :

Таблица 3 – код репозитория обрабатывающий запросы для класса User.

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, Long> {  User findByEmail(String email);  User findByEmailAndPassword(String email, String password);  Page<User> findAll(Pageable pageable); } |

Пример на репозиторий для обработке SQL запросов для класса Message:

Таблица 4 – код репозитория обрабатывающий запросы для класса Message.

|  |
| --- |
| public interface MessageRepository extends JpaRepository<Message, Long> {} |

Пример на репозиторий для обработке SQL запросов для класса Conversation:

Таблица 5 – код репозитория обрабатывающий запросы для класса Conversation.

|  |
| --- |
| public interface ConversationRepository extends JpaRepository<Conversation, Long> {} |

Пример на репозиторий для обработке SQL запросов для класса MidConversation:

Таблица 6 – код репозитория обрабатывающий запросы для класса MidConversation.

|  |
| --- |
| public interface MidConversationRepository extends JpaRepository<MidConversation, Long> {} |

**2.5 Краткое сравнение ролей слоёв:**

| **Слой** | **Аннотация** | **Ответственность** |
| --- | --- | --- |
| Контроллер | @RestController | Обработка HTTP-запросов, маршрутизация |
| Сервис | @Service | Бизнес-логика, правила, транзакции |
| Репозиторий | @Repository | Доступ к данным, работа с базой данных |

### ****2.6 Итоговая архитектура Spring Boot приложения:****

[ HTTP-запрос ]

↓

@RestController (Контроллер)

↓

@Service (Сервис / бизнес-логика)

↓

@Repository (Репозиторий / база данных)

↓

[ База данных ]

**3. Проектирование системы**

**3.1 Структура проекта**

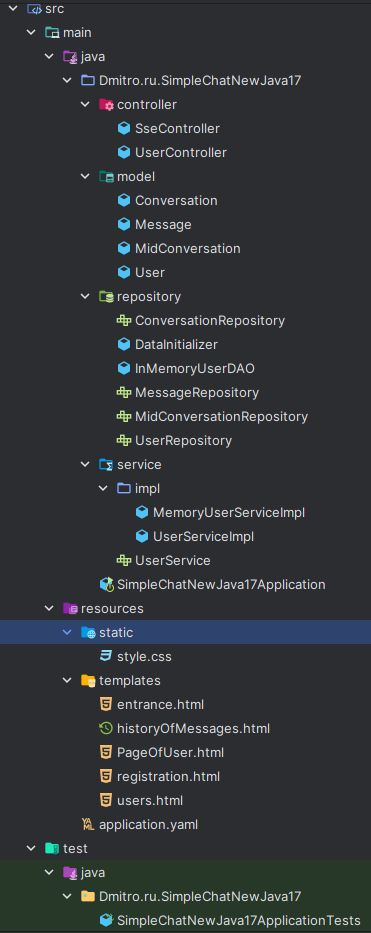


Рисунок 1 – Структура проекта.

**3.2 База данных**

**3.2.1 Список таблиц с полями и типами данных**

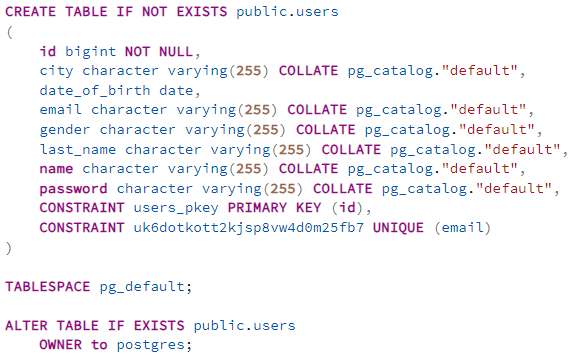


Рисунок 2 – создание таблицы users на SQL.

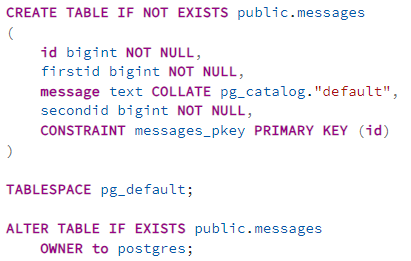


Рисунок 3 – создание таблицы messages на SQL.

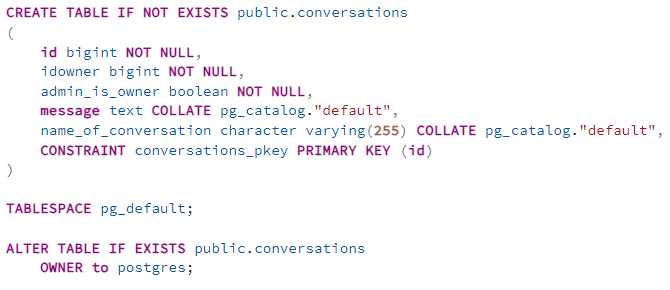


Рисунок 4 – создание таблицы conversations на SQL.

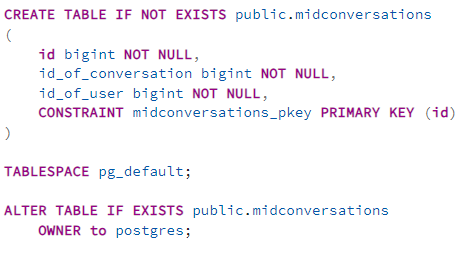


Рисунок 5 – создание таблицы midconversations на SQL.

**3.2.2 ORM-модели**

В этой системе для взаимодействия с базой данных используется технология JPA (Java Persistence API). Каждая таблица в базе данных представлена как сущность в коде, аннотированная с помощью @Entity. Эти классы отражают структуру данных, с которыми работает система.

Пример на модель типа User:

Таблица 7 – код модели типа User.

|  |
| --- |
| @Data @Entity @Table(name = "users") public class User {  public User(){}  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  private long id;  private String name;  @Column(name = "last\_name")  private String lastName;  @Column(name = "date\_of\_birth")  private LocalDate dateOfBirth;  private String gender;  private String city;  @Column(unique = true)  private String email;  private String password;  @Transient  private int age;  public int getAge() {  if (dateOfBirth == null) {  return 0; // или другое значение по умолчанию  }  return Period.*between*(dateOfBirth, LocalDate.*now*()).getYears();  } } |

Пример на модель типа Message:

Таблица 8 – код модели типа Message.

|  |
| --- |
| @Data @Entity @Table(name = "messages") public class Message {  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  private long id;  @Lob  @Column(columnDefinition = "TEXT")  private String message;  private long firstID;  private long secondID; } |

Пример на модель типа Conversation:

Таблица 9 – код модели типа Conversation.

|  |
| --- |
| @Data @Entity @Table(name = "conversations") public class Conversation {  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  private long id;  @Lob @Column(columnDefinition = "TEXT")  private String message;  private String nameOfConversation; private long IDOwner;  private boolean adminIsOwner;  } |

Пример на модель типа MidConversation:

Таблица 10 – код модели типа MidConversation.

|  |
| --- |
| @Data @Entity @Table(name = "midconversations") public class MidConversation {  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  private long id;  private long IdOfUser;  private long IdOfConversation; } |

* Аннотация @Entity указывает, что этот класс является сущностью, которая будет связана с таблицей в базе данных.
* @Id — ключевое поле, идентификатор записи.
* @GeneratedValue — указывает, как генерировать значения для первичного ключа.
* @Table используется для указания имени таблицы, которая будет связана с сущностью.
* @Column — Указание колонок в таблице базы данных.
* @Lob — Большие объекты (Large Object).
* @Transient — Исключение поля из сохранения в базе данных.

**3.3 Технологический стек**

* Spring Boot
* Hibernate
* PostgreSQL
* Maven
* Lombok
* Web
* Data-jpa
* Test
* Thymeleaf
* Devtools
* Jackson-datatype-jsr310
* Sqlite-jdbc
* Mysql-connector-java

**3.4 Паттерны проектирования в разработке системы**

1. Builder

* Помогает создавать сложные объекты с множеством параметров, улучшая читаемость и удобство создания.

1. Repository

* Абстрагирует доступ к данным, скрывая детали взаимодействия с базой данных, и предоставляет удобные методы для работы с сущностями.

1. Service Layer

* Организует бизнес-логику приложения в отдельный слой, отделяя её от контроллеров и упрощая управление транзакциями.

1. Observer

* Позволяет уведомлять несколько объектов о произошедших событиях, обеспечивая асинхронное оповещение об изменениях.

1. Transactional

* Обеспечивает управление транзакциями, гарантируя, что операции с данными либо полностью завершаются, либо откатываются при ошибке.

1. DTO (Data Transfer Object)

* Используется для передачи данных между различными слоями системы, минимизируя количество передаваемых данных и избегая избыточных зависимостей.

1. Singleton

* Обеспечивает наличие только одного экземпляра класса, который используется по всему приложению, обеспечивая централизованный доступ.

1. Factory

* Предоставляет интерфейс для создания объектов, делая процесс создания сложных объектов независимым от их конкретных типов.

1. Facade

* Предоставляет упрощённый интерфейс для сложной подсистемы, скрывая её детали и облегчая взаимодействие с ней.

1. Strategy

* Позволяет изменять поведение объекта в зависимости от выбранной стратегии, делая систему гибкой и расширяемой.

1. Command

* Инкапсулирует запрос как объект, позволяя передавать параметры и выполнить запрос позже, например, для реализации отмены операций или очереди запросов.

1. MVC (Model-View-Controller)

* Использование RestController в проекте

**3.5 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.

Таблица 11.1 – код тестов (часть 1)

|  |
| --- |
| @ExtendWith(MockitoExtension.class) class SimpleChatNewJava17ApplicationTests {  @Mock  private UserService userService;  @Mock  private Model model;  @Mock  private HttpSession session;  @InjectMocks  private UserController userController; // Замените на ваш контроллер  // метод тестирует авторизацию  @Test  public void testEntranceByEmail\_SuccessfulLogin() {  // Arrange  String email = "Danil227\_1743351255984@mail.ru";  String password = "2fsae22fd";  User realUser = new User();  realUser.setEmail(email);  realUser.setPassword(password);  *when*(userService.FindUserByEmailAndPassword(email, password)).thenReturn(realUser);  // Act  String viewName = userController.EntranceByEmail(email, password, model, session);   // Assert  *assertEquals*("entrance", viewName);   *verify*(model, *never*()).addAttribute(*eq*("error"), *anyString*());  } |

Продолжение на следующей странице…

Таблица 11.2 – код тестов (часть 2)

|  |
| --- |
| // метод тестирует регистрацию с неверным проверочным паролем  @Test  void registration\_PasswordMismatch\_ShouldReturnRegistrationView() {  // Arrange  User newUser = new User();  newUser.setName("Nik");  newUser.setGender("male");  newUser.setCity("Navashino");  newUser.setLastName("Bor");  newUser.setAge(18);  newUser.setEmail("new@example.com");  newUser.setPassword("securePass");  String testPassword = "differentPassword";  // Act  String result = userController.Registration(newUser, model, session, testPassword);  // Assert  *assertEquals*("registration", result);  *verify*(model).addAttribute("error", "Пароли не сходяться!");  *verifyNoInteractions*(session);  } // метод тестирует полную регистрацию  @Test  void registration\_NewUser\_ShouldSaveAndReturnEntrance() {  // Arrange  User newUser = new User();  newUser.setName("Nik");  newUser.setGender("male");  newUser.setCity("Navashino");  newUser.setLastName("Bor");  newUser.setAge(18);  newUser.setEmail("new@examplee.com");  newUser.setPassword("securePass");  System.*out*.println("EMAIL in test: " + newUser.getEmail());  // Пользователь НЕ найден — значит, он новый  *when*(userService.FindUserByEmail("new@examplee.com")).thenReturn(null);  // Act  String result = userController.Registration(newUser, model, session, "securePass"); |

Продолжение на следующей странице…

Таблица 11.3 – код тестов (часть 3)

|  |
| --- |
| // Assert  *assertEquals*("entrance", result);  *verify*(userService).SaveUser(newUser);  *verify*(userService).FindUserByEmail("new@examplee.com");  *verify*(session).setAttribute("user", newUser);  *verify*(model).addAttribute("newUser", newUser);  }  // Тестируем проверку на сходие email  @Test  void registration\_EmailAlreadyRegistered\_ShouldReturnRegistrationWithError() {  // Arrange  User newUser = new User();  newUser.setName("Nik");  newUser.setGender("male");  newUser.setCity("Navashino");  newUser.setLastName("Bor");  newUser.setAge(18);  newUser.setEmail("Danil227\_1743351255984@mail.ru");  newUser.setPassword("securePass"); w*hen*(userService.FindUserByEmail("Danil227\_1743351255984@mail.ru")).thenReturn(new User());  // Act  String result = userController.Registration(newUser, model, session, "securePass");  // Assert  *assertEquals*("registration", result);  *verify*(model).addAttribute("error", "Такой email уже зарегистрирован!");  *verify*(userService, *never*()).SaveUser(*any*());  *verify*(session, *never*()).setAttribute(*eq*("user"), *any*());  } // протестировали поиск  @Test  void testFindUserByEmail\_NoUsersFound\_ShouldAddErrorAndAttributesToModel() {  // Arrange  String email = "test@examplee.com";  int page = 0;  int size = 12;  boolean sort = false;  long idOfUserActual = 5; |

Продолжение на следующей странице…

Таблица 11.4 – код тестов (часть 4)

|  |
| --- |
| // Создаем пустой список, так как в тесте мы имитируем ситуацию, когда пользователь не найден  List<User> findOfUsers = new ArrayList<>();   // Создаем пустую страницу пользователей (поскольку findOfUsers пуст)  Page<User> usersPage = new PageImpl<>(new ArrayList<>());   // Создаем актуального пользователя  User actualUser = new User();  actualUser.setId(5L);  actualUser.setEmail("test@examplee.com");   // Act  String viewName = userController.FindUserByEmail(page, size, email, sort, idOfUserActual, model, session);   // Assert  *assertEquals*("users", viewName); // Ожидаем, что вернется представление users// Проверяем размер списка для добавления в разговор (будет 0)  }   @Test  void testInputMessages() {  String message = "Hello world";  long idOfUserActual = 10;  User actualUser = new User();  actualUser.setId(idOfUserActual);  actualUser.setEmail("test@examplee.com");  int id = 5;  Message newMessage = new Message();  User companion = new User();  companion.setId(id);  companion.setEmail("companion@example.com");  newMessage.setFirstID(actualUser.getId());  newMessage.setSecondID(companion.getId());  newMessage.setMessage(actualUser.getId() + " " + message + '\n');   String viewName = userController.InputMessages(message, id, idOfUserActual, model, session);  *assertEquals*("PageOfUser", viewName);  } |

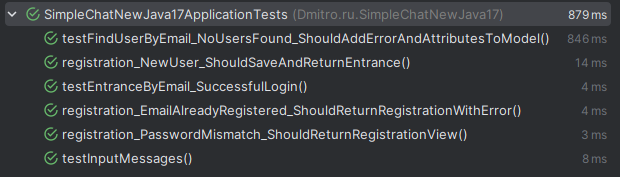


Рисунок 6 – успешная работа тестов.

**4 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.

Проверка программы представлена в рисунках 7-26.

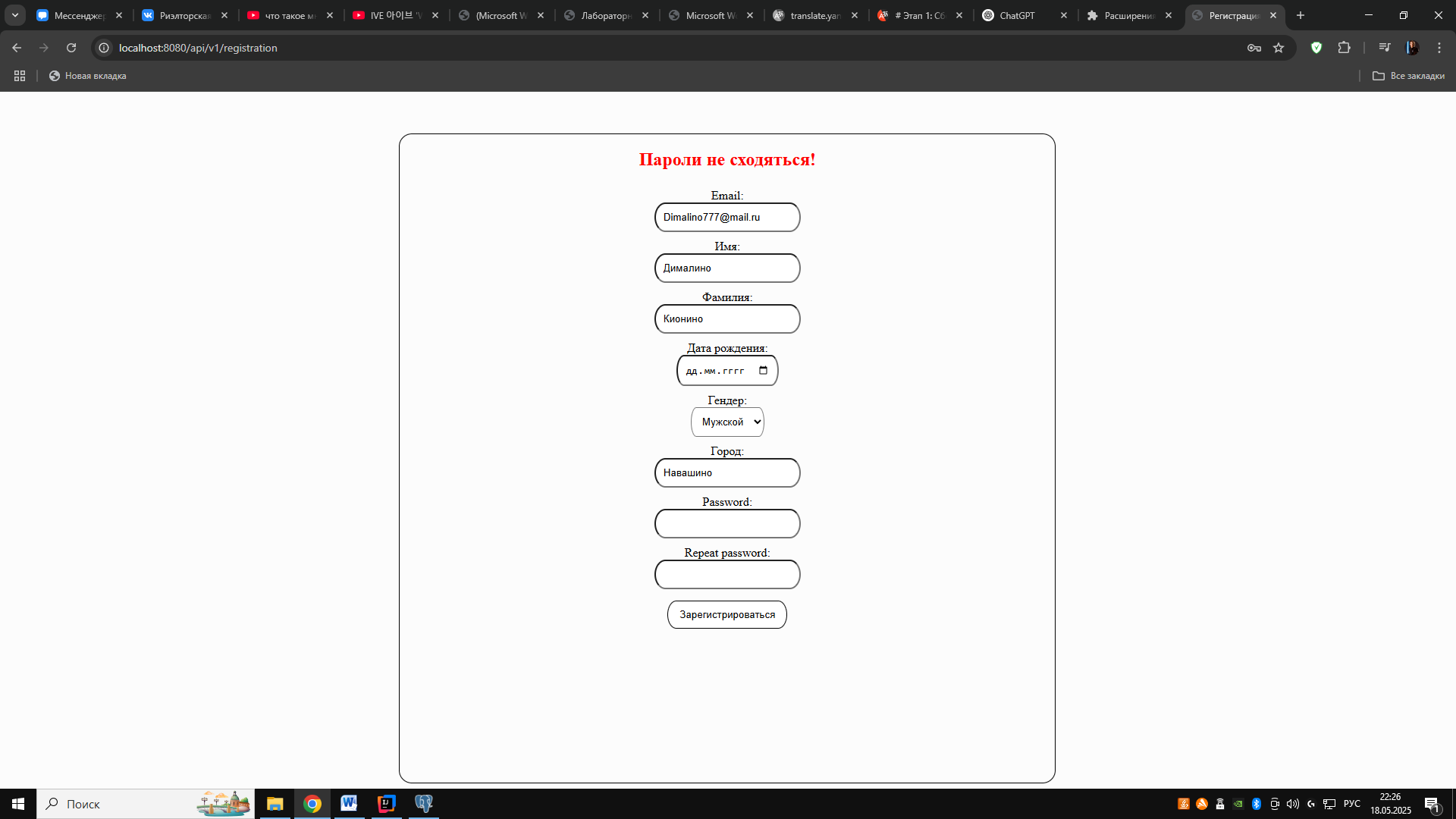


Рисунок 7 – успешная защита при регистрации от неправильного ввода проверочного пароля.

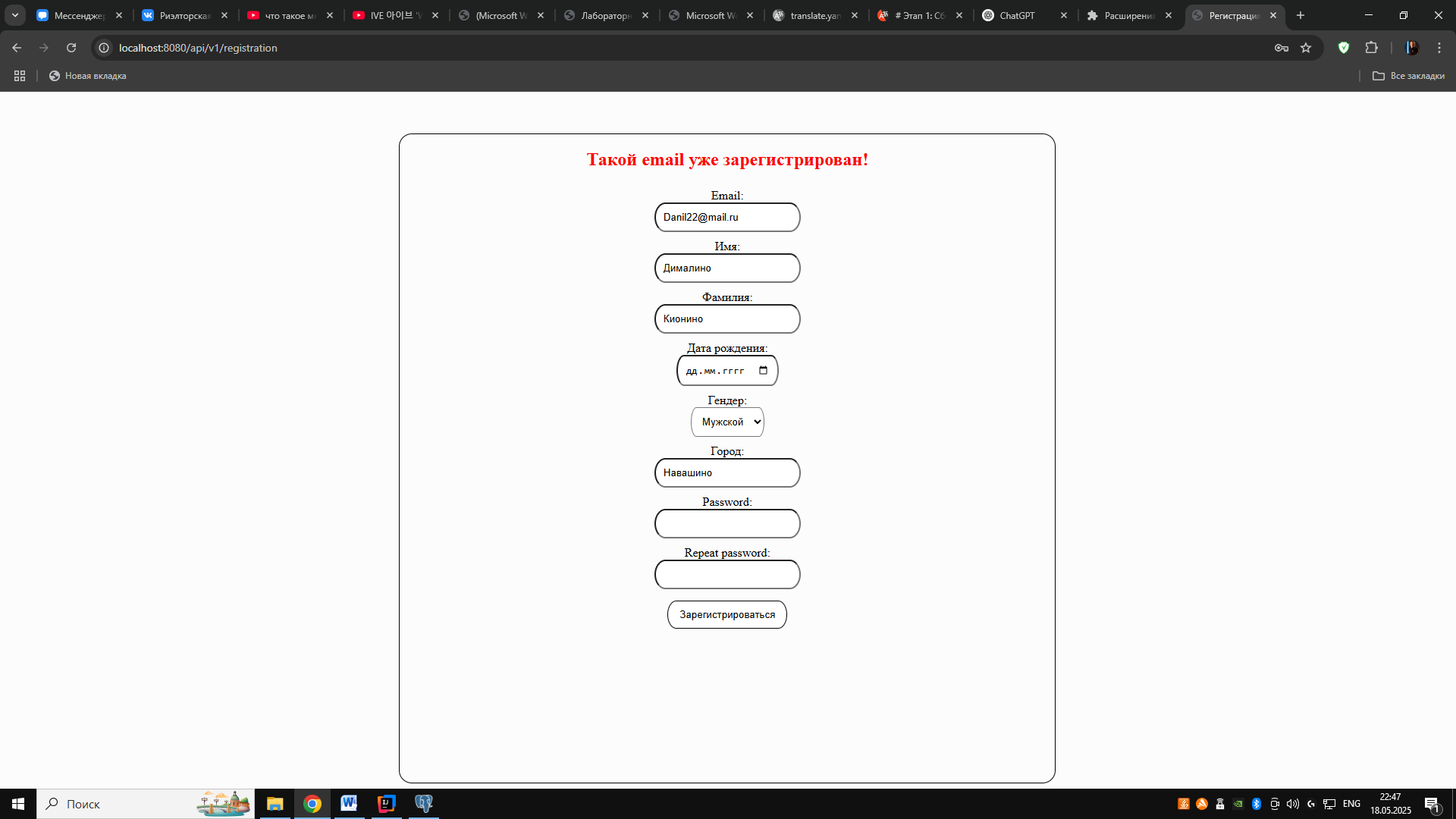


Рисунок 8 – успешная защита при регистрации от ввода существующего email.

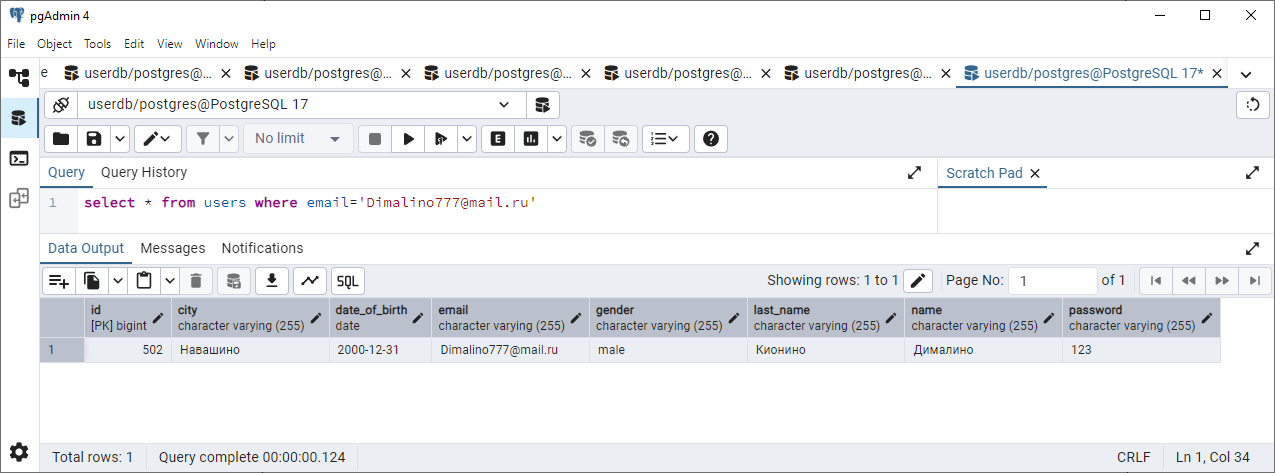


Рисунок 9 – успешный ввод данных в базу данных при правильной регистрации.

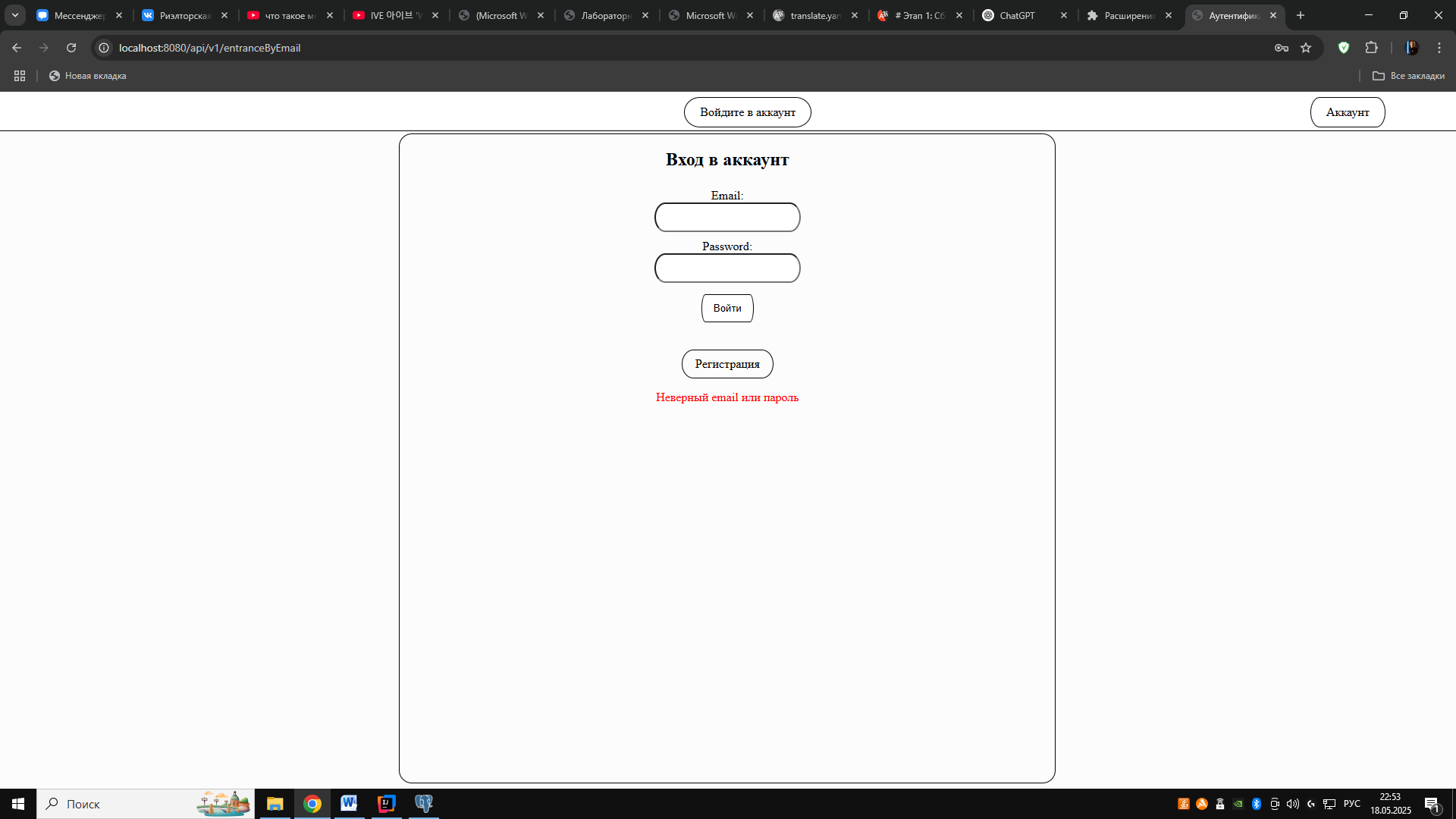


Рисунок 10 – успешная защита при аутентификации пользователя при неправильном вводе пароля или email.

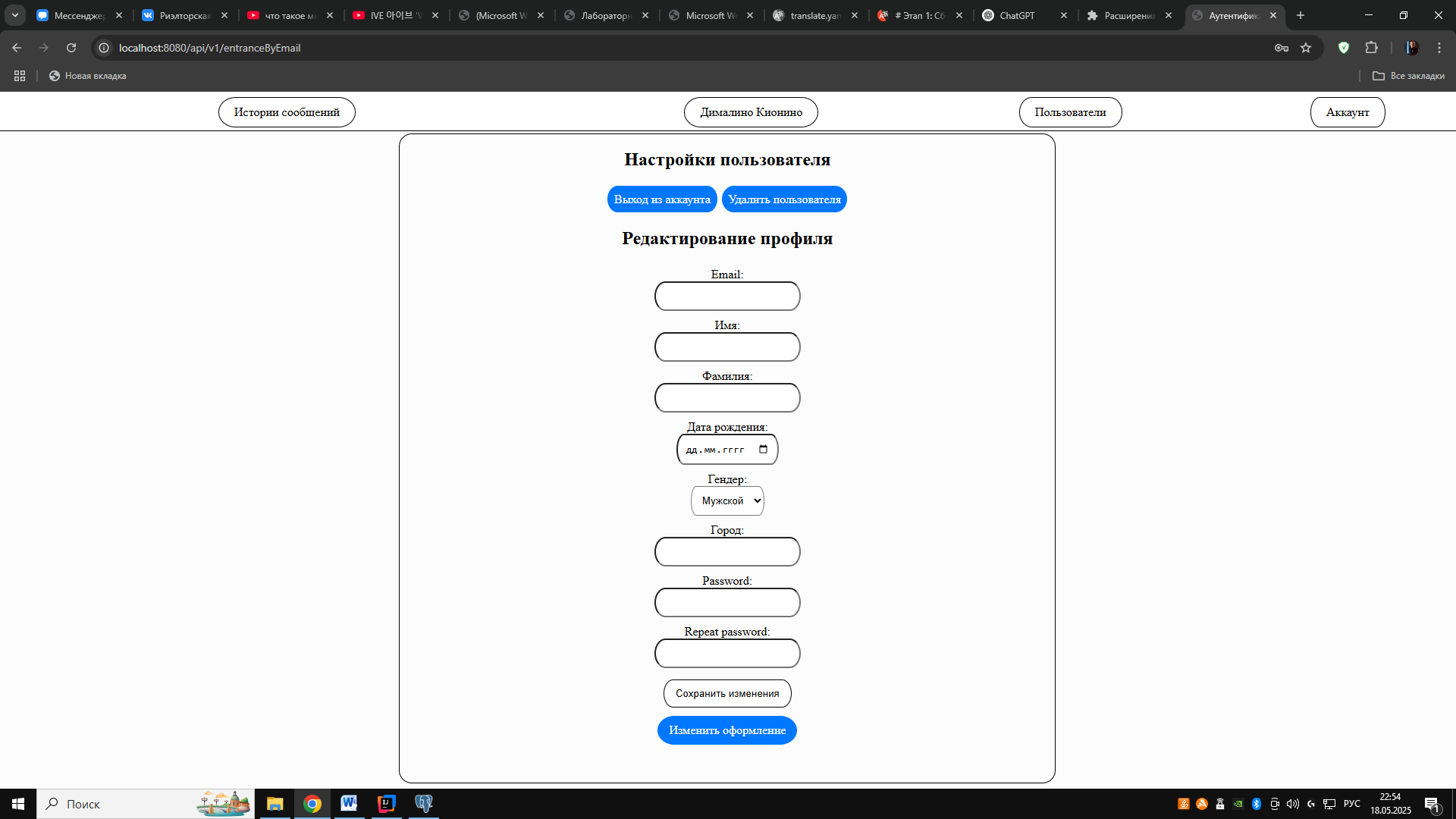


Рисунок 11 – успешная аутентификация пользователя.

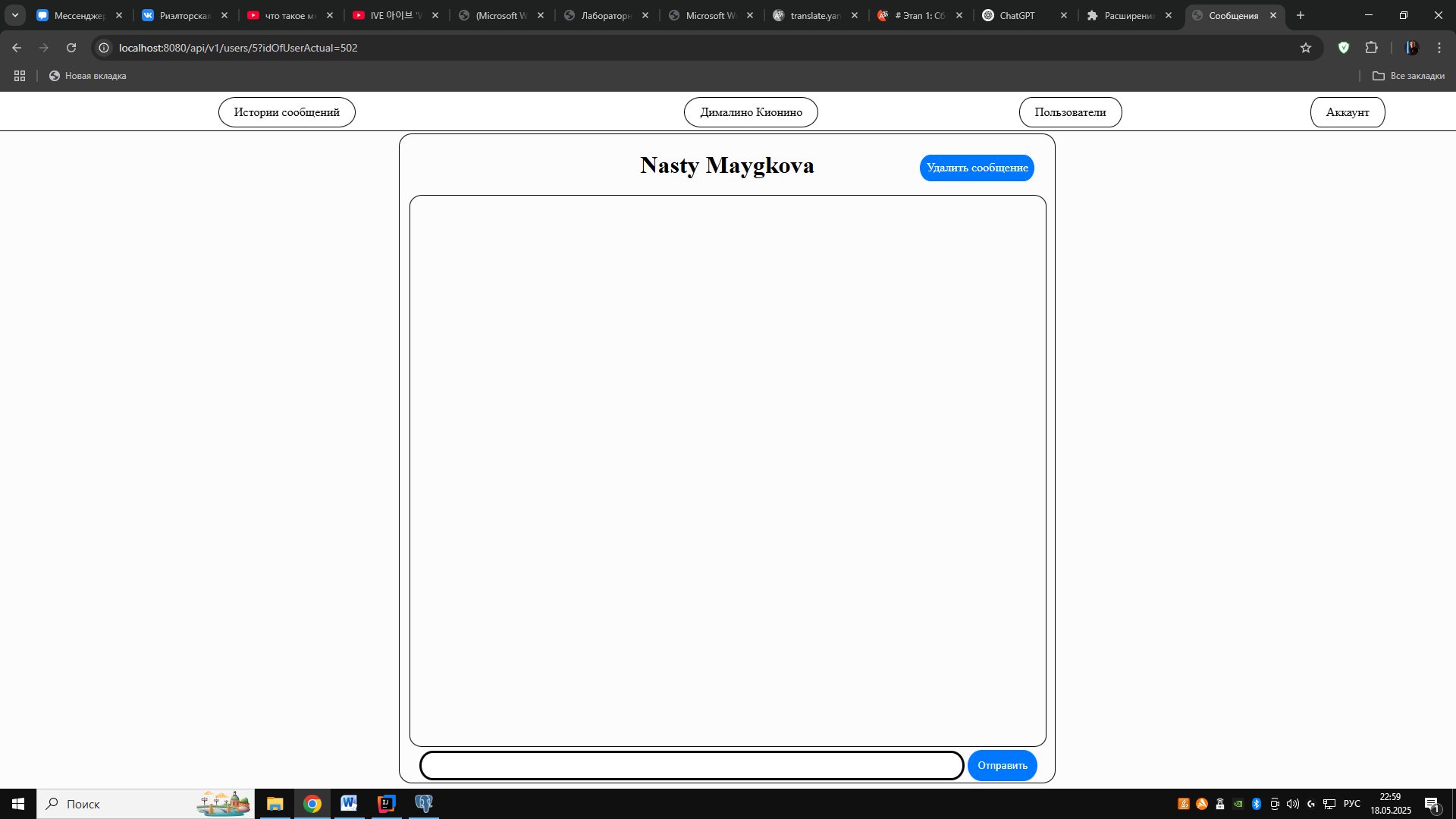


Рисунок 12 – успешное открытие сообщений с пользователем.

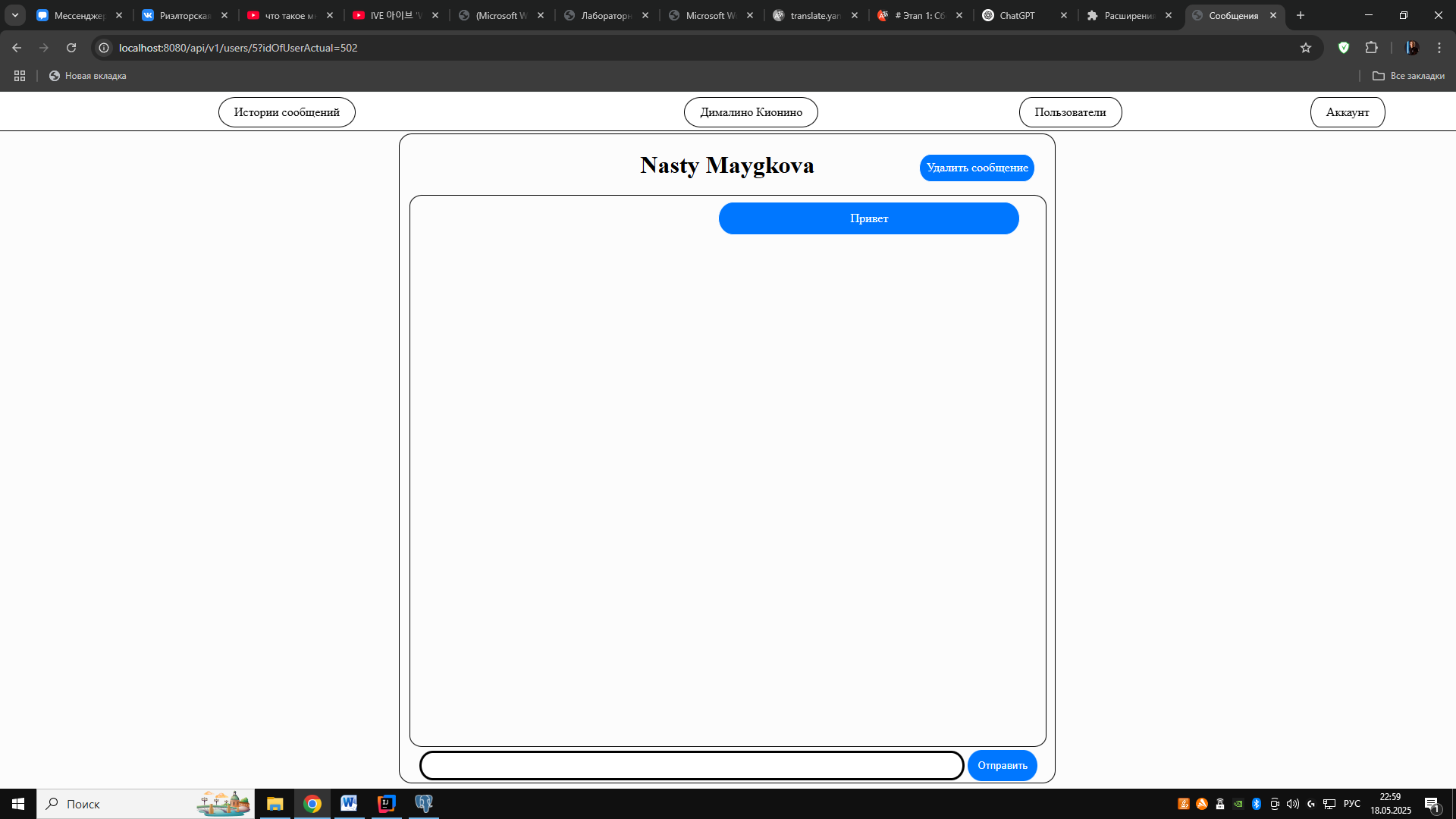


Рисунок 13 – успешно отправил сообщений пользователю.

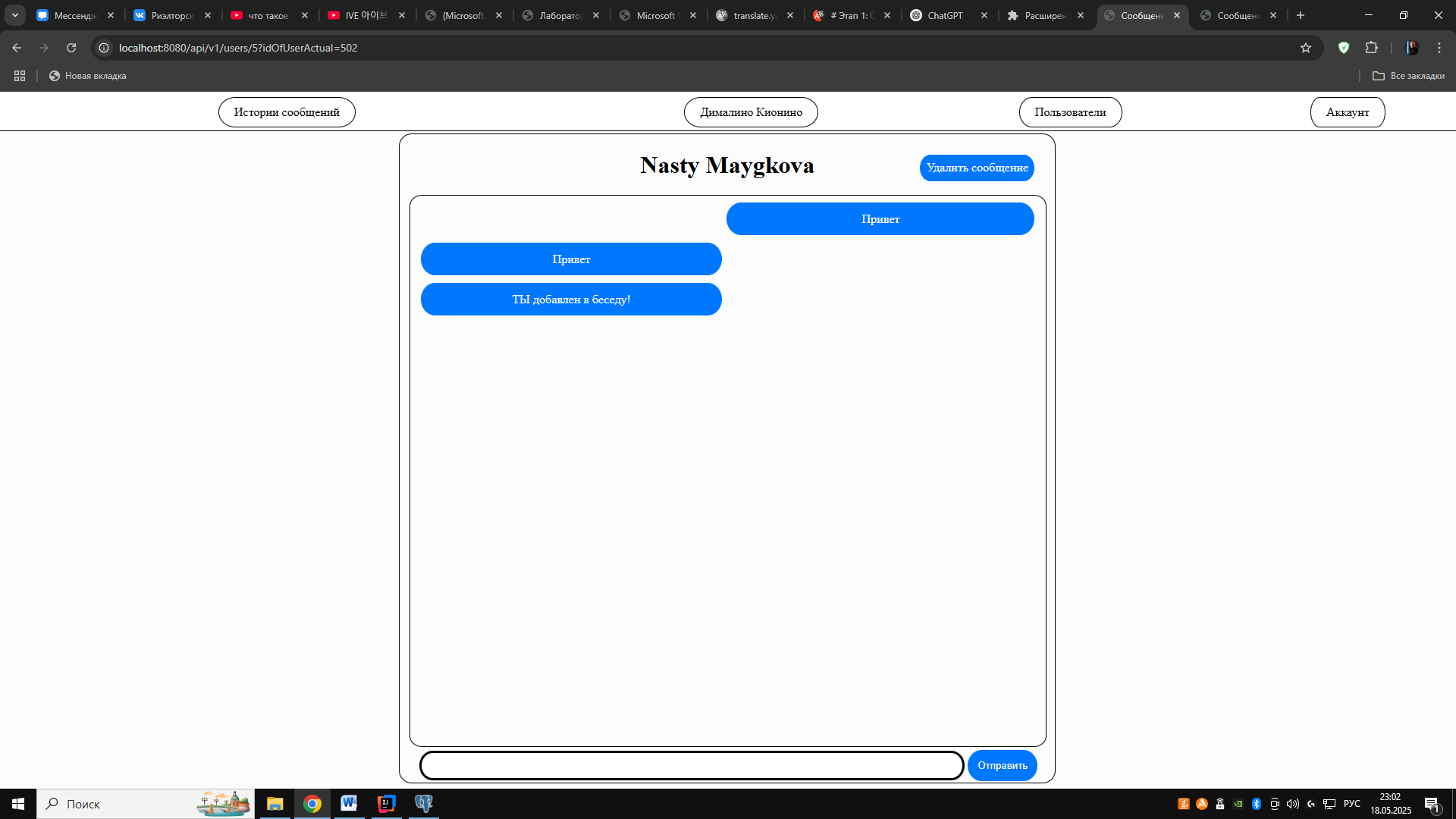


Рисунок 14 – успешно получил сообщение от пользователя.

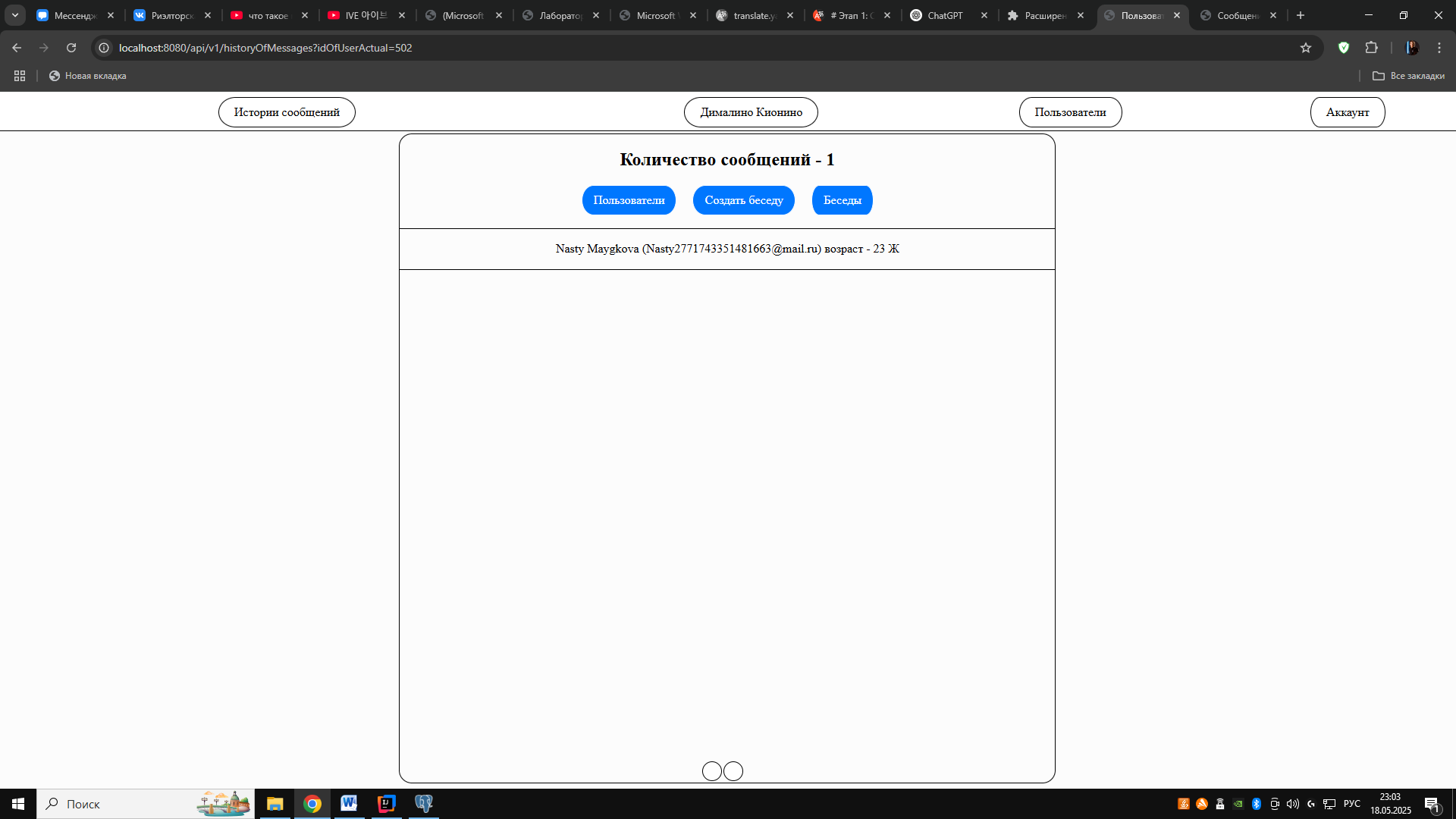


Рисунок 15 – успешно открыл историю переписок с пользователями.

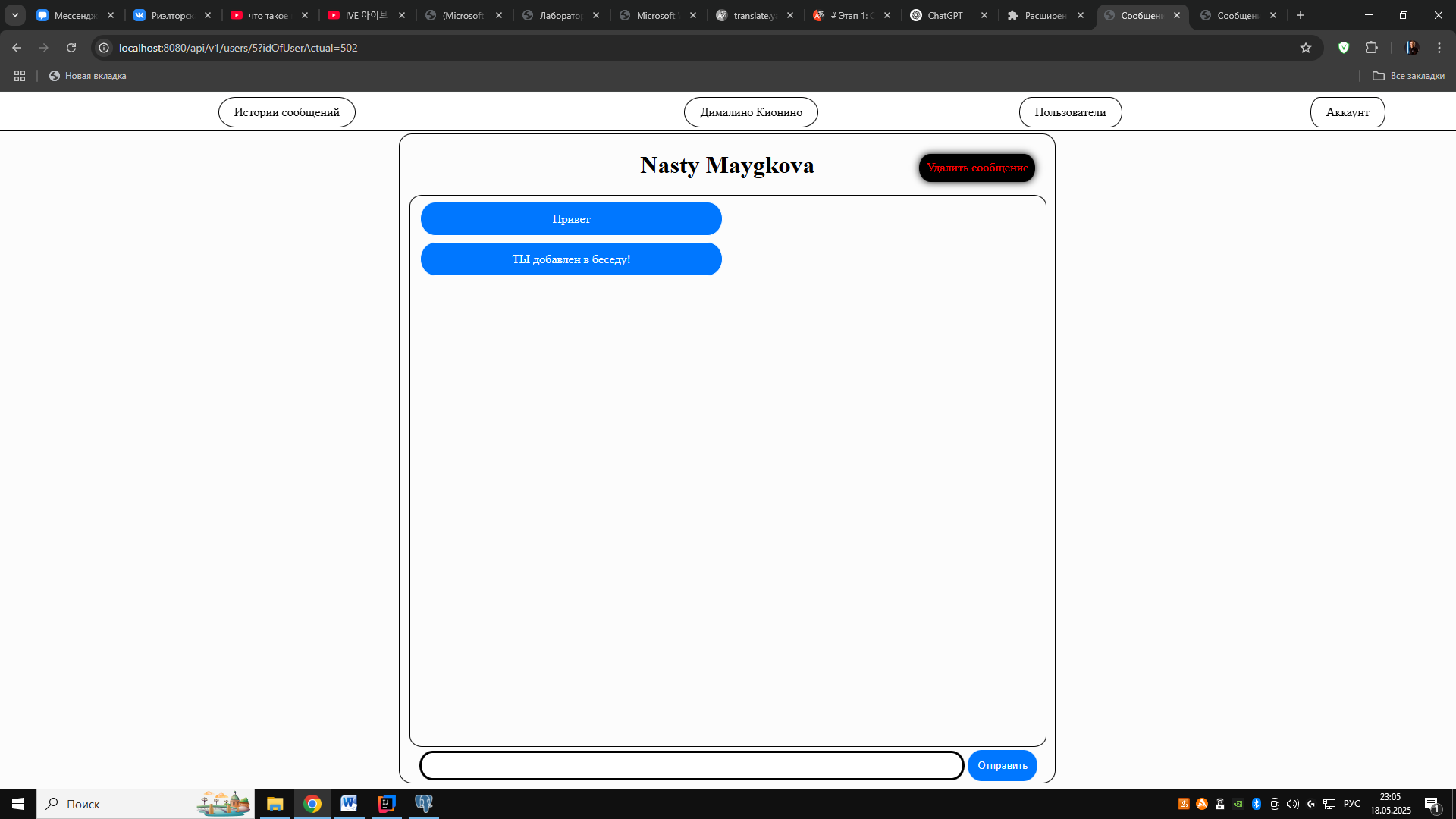


Рисунок 16 – успешно удалил своё первое сообщение.

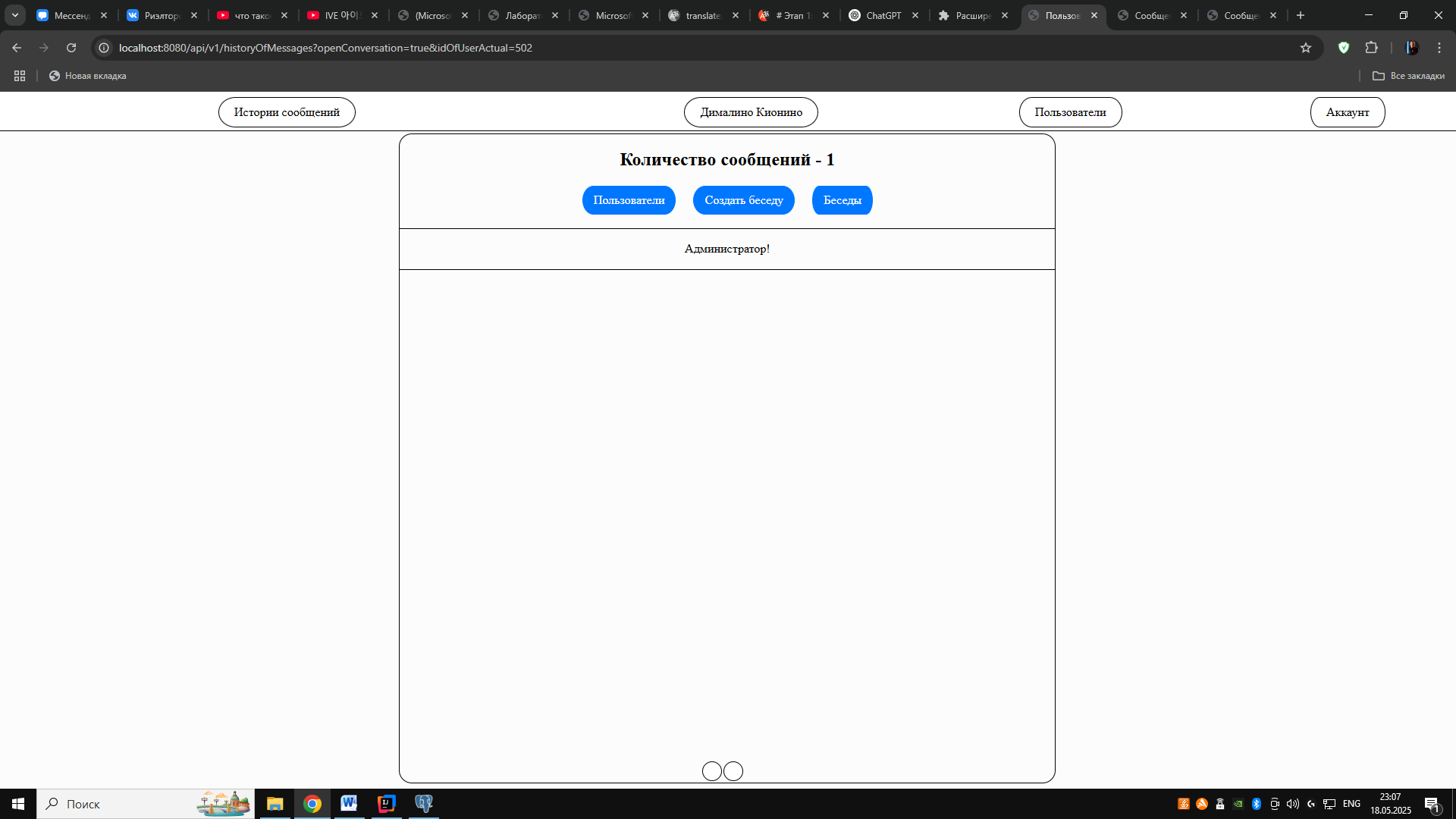


Рисунок 17 – успешно отрыл историю бесед, в который состоит мой пользователь.

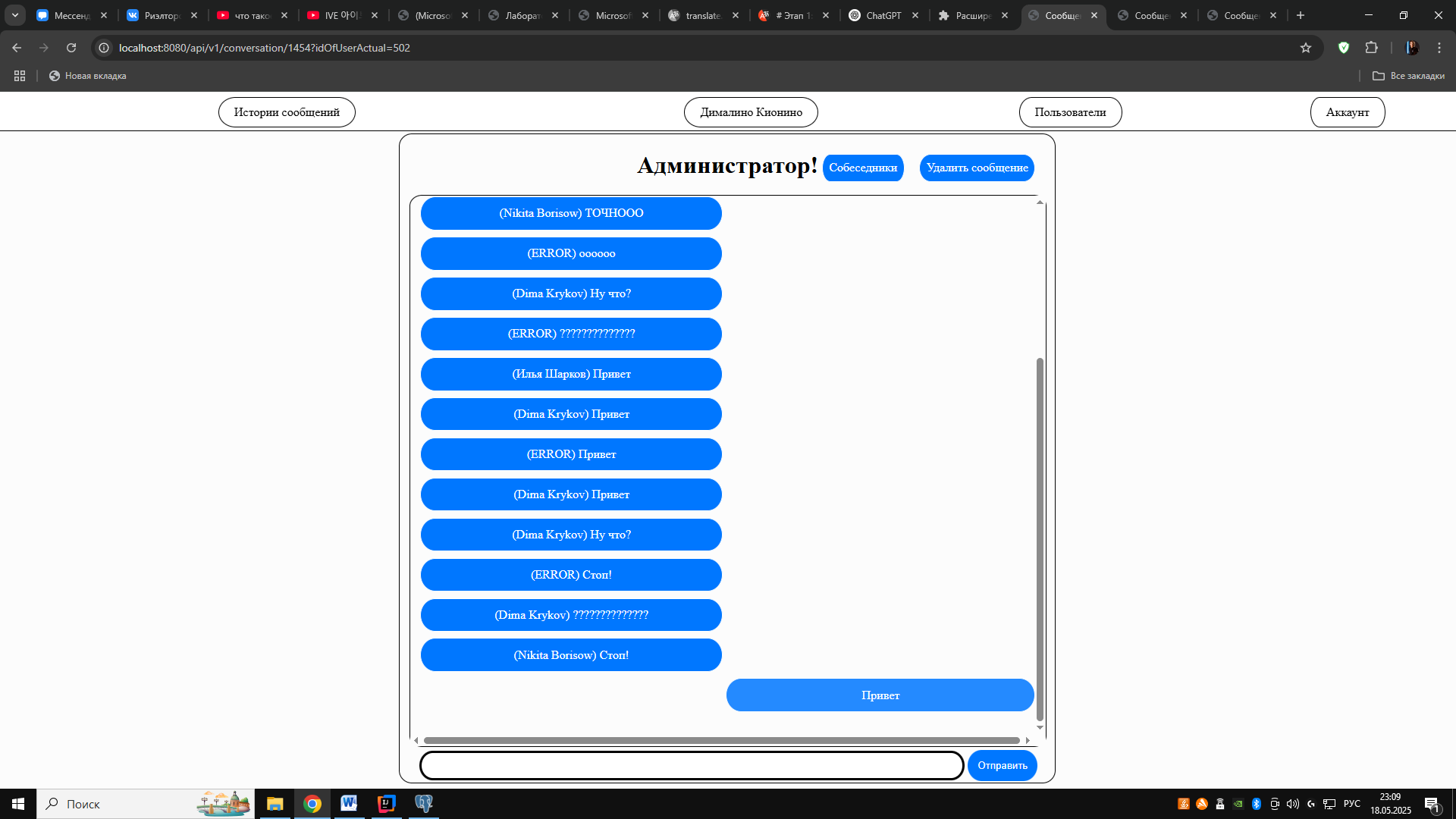


Рисунок 18 – успешно отправил сообщение в беседу.

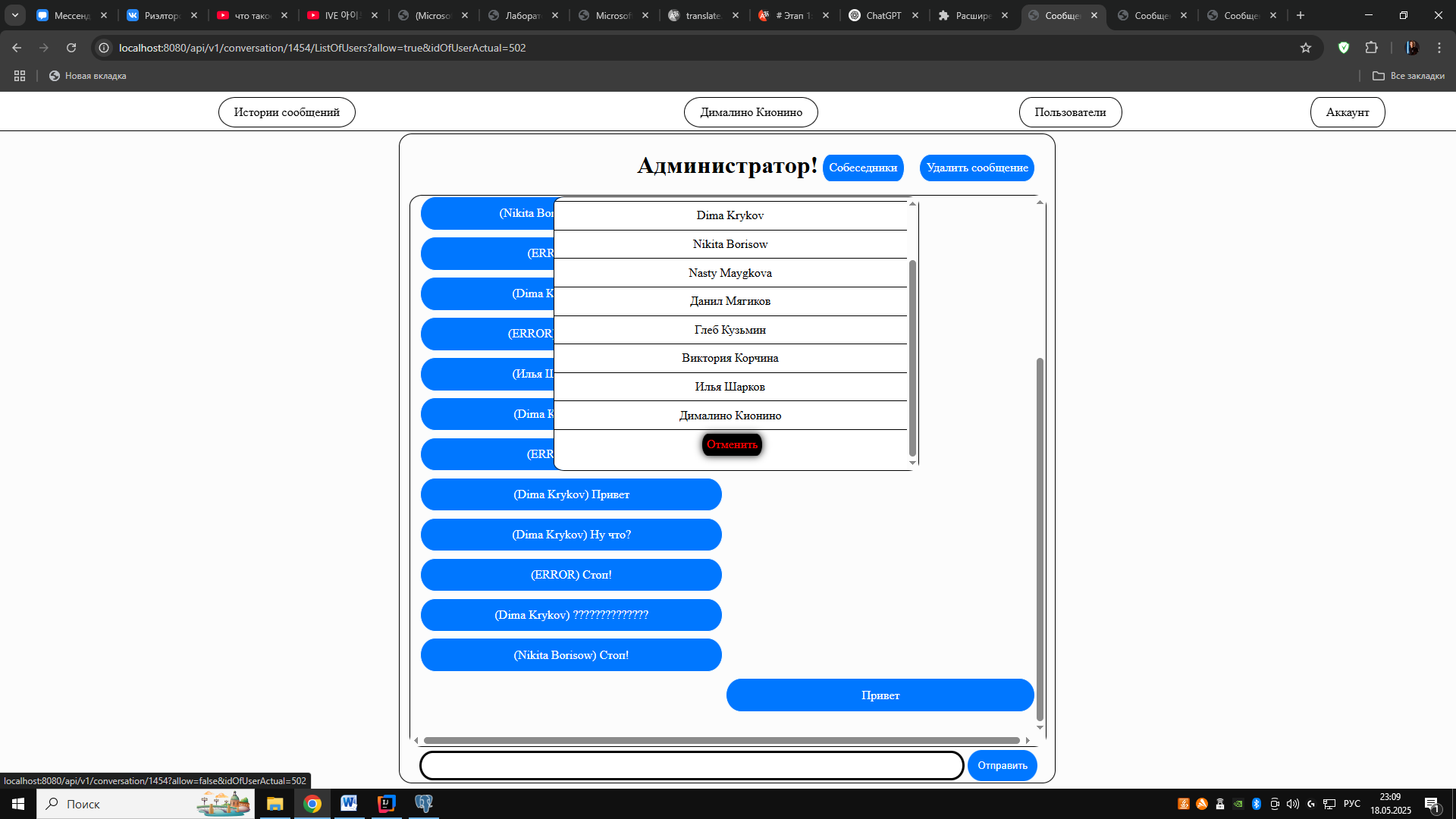


Рисунок 19 – успешно проверил список пользователей состоящих в беседе, а так же ограничение на добавление новых пользователей.

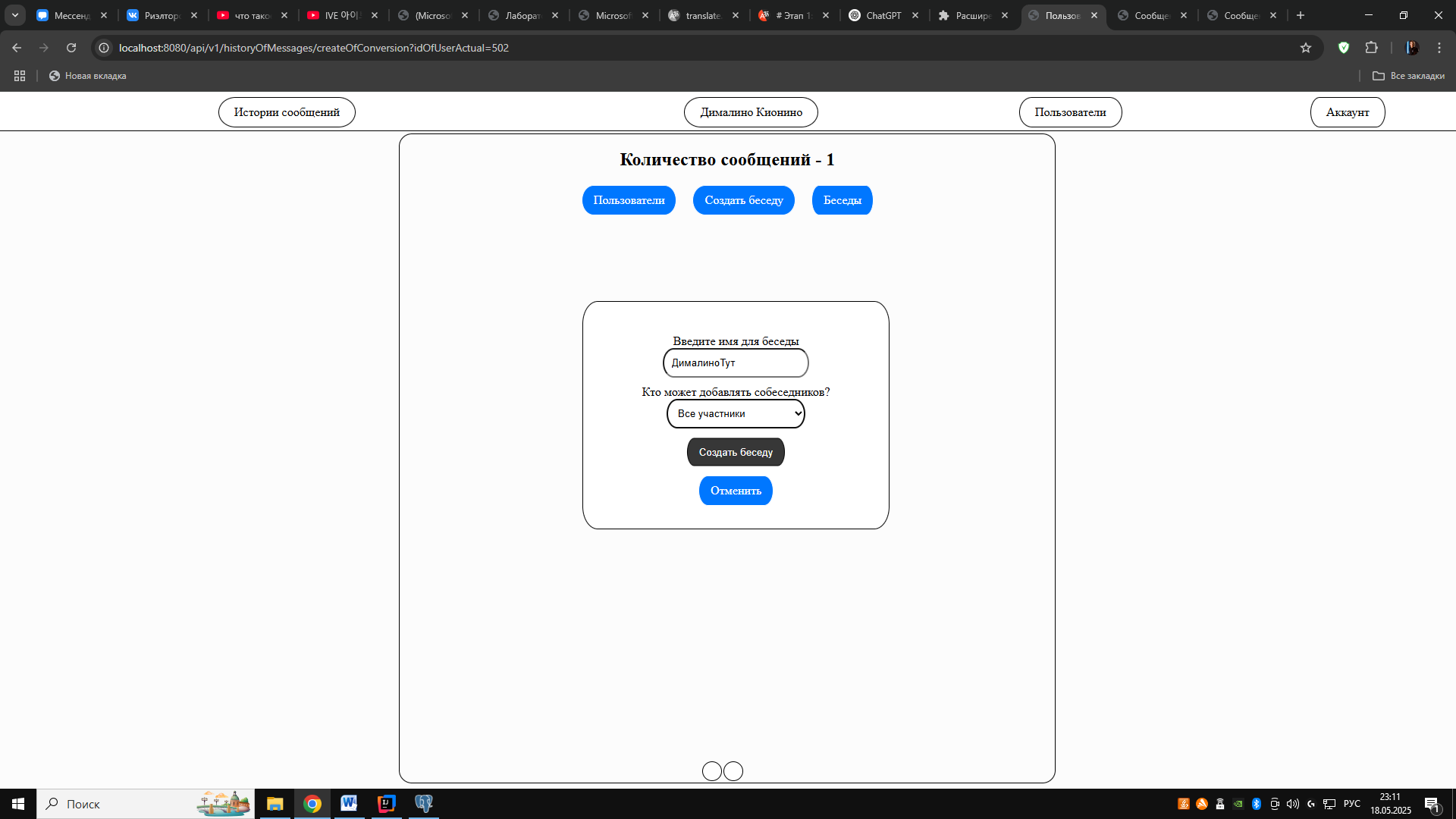


Рисунок 19 – успешно открыл окно с созданием беседы.

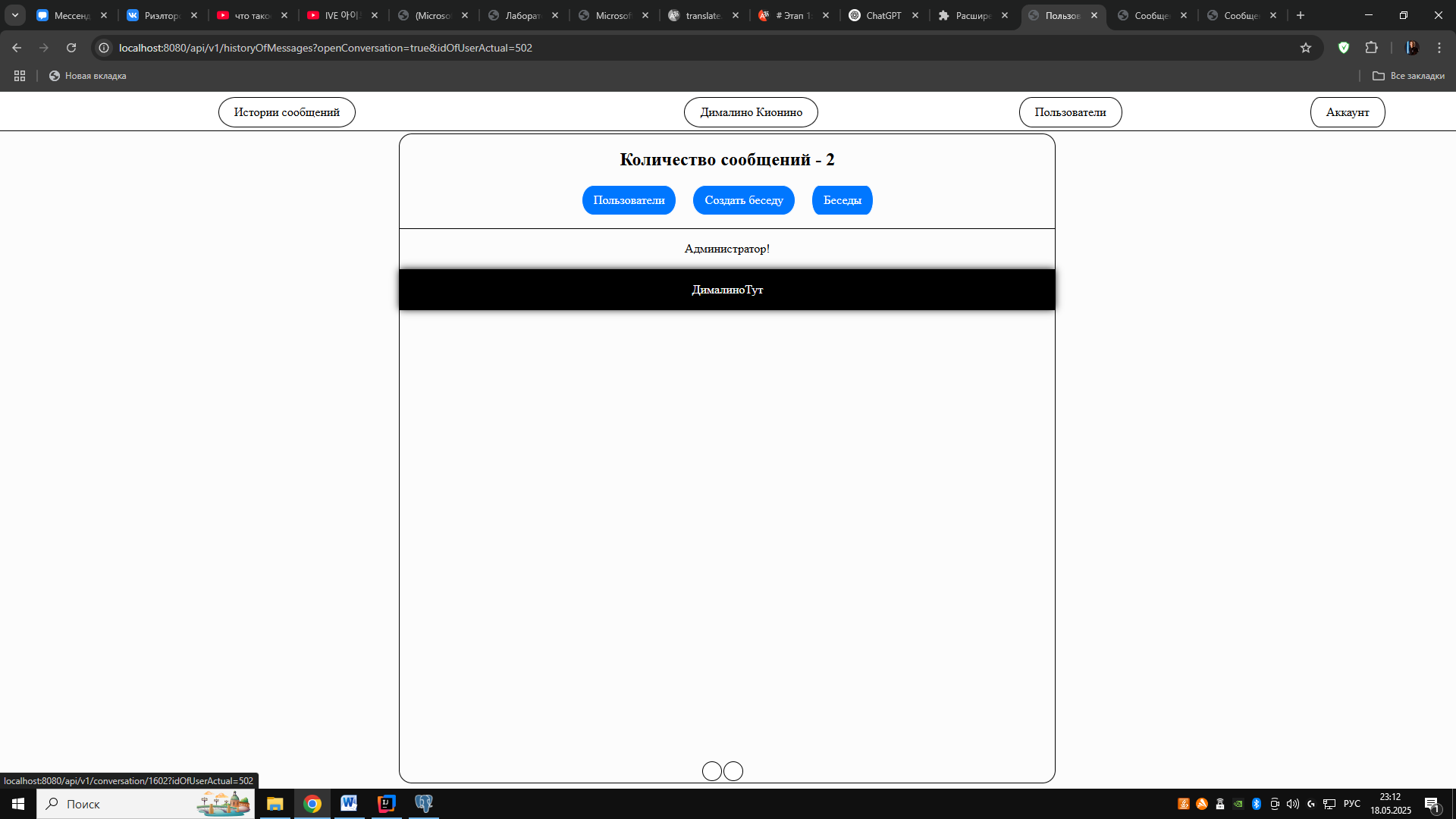


Рисунок 20 – успешно создал новую беседу.

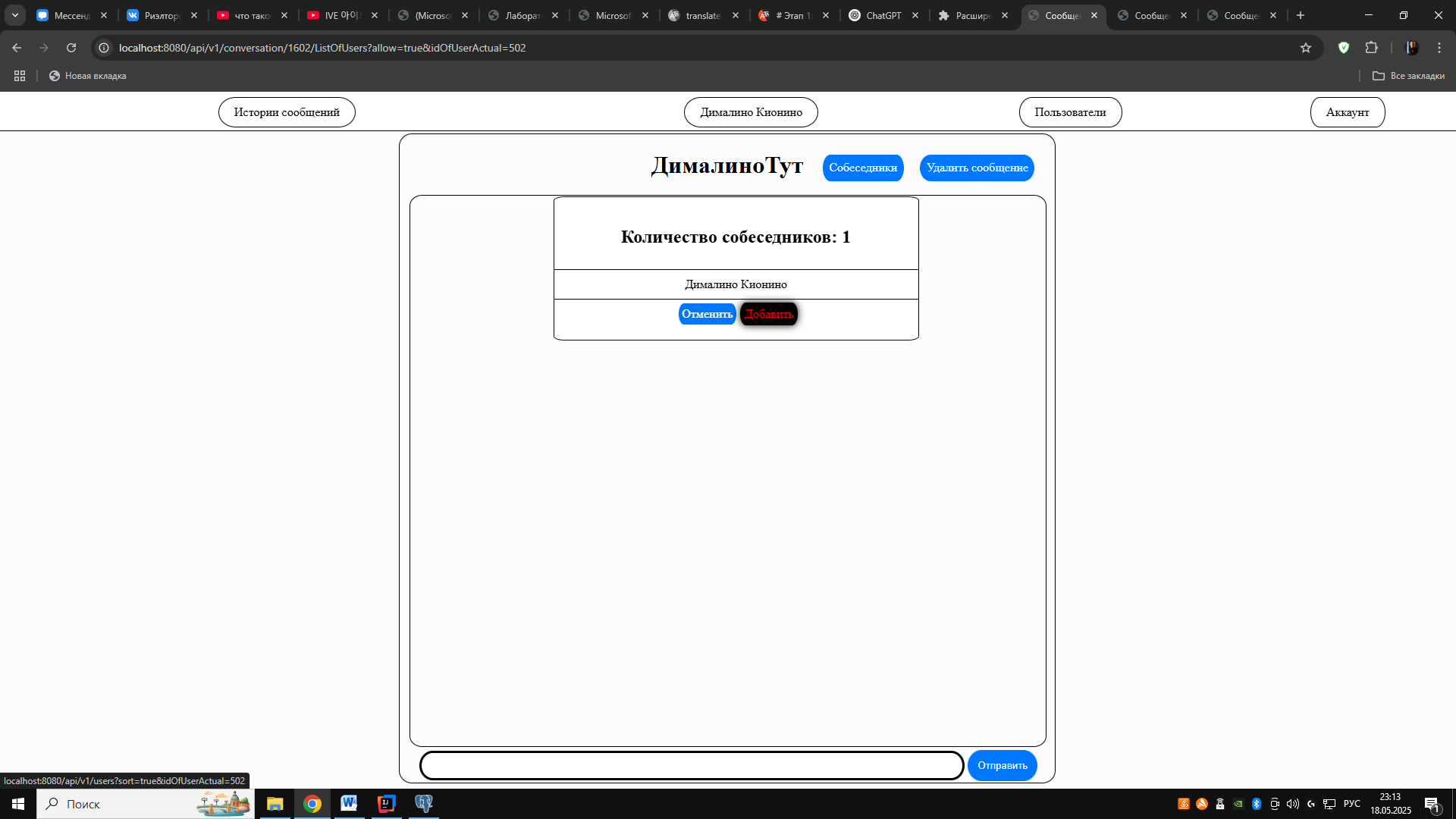


Рисунок 21 – успешно открыл список пользователей находящих в беседе и проверил возможность для добавления новых пользователей в беседу.

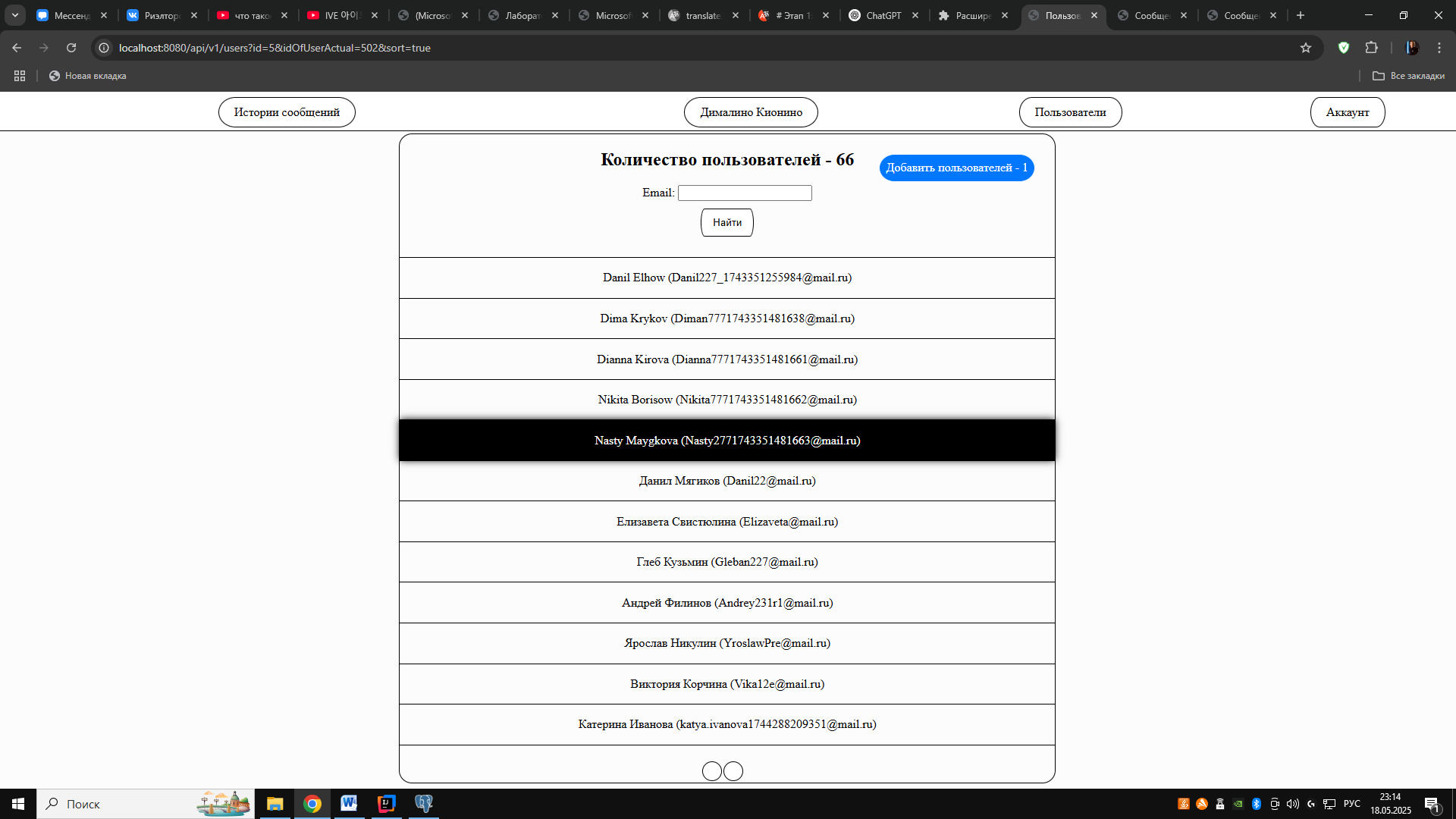


Рисунок 22 – успешно кликнул на пользователя для добавления его в свою беседу.

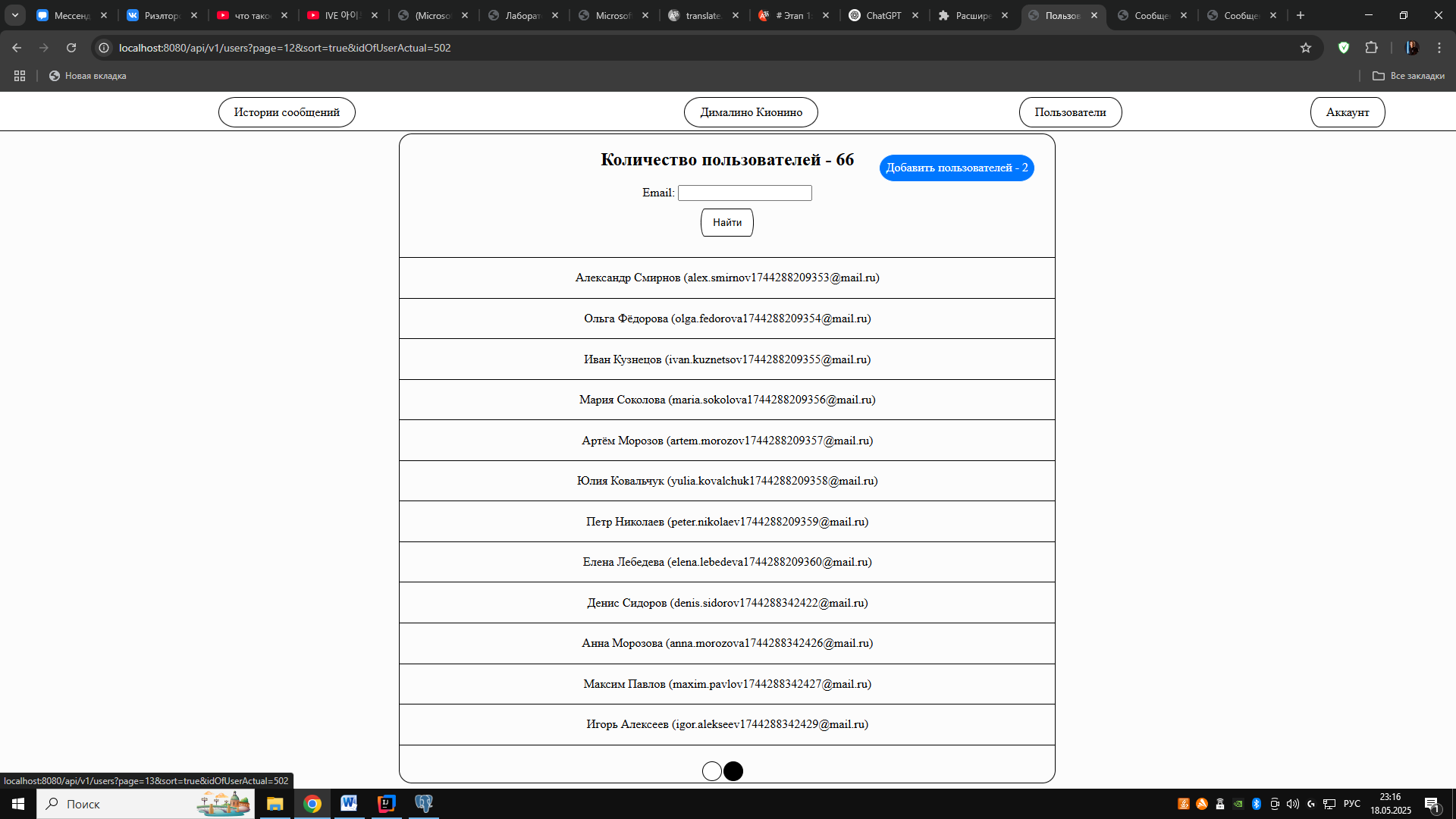


Рисунок 23 – успешно протестировал пагинацию для списка пользователей.

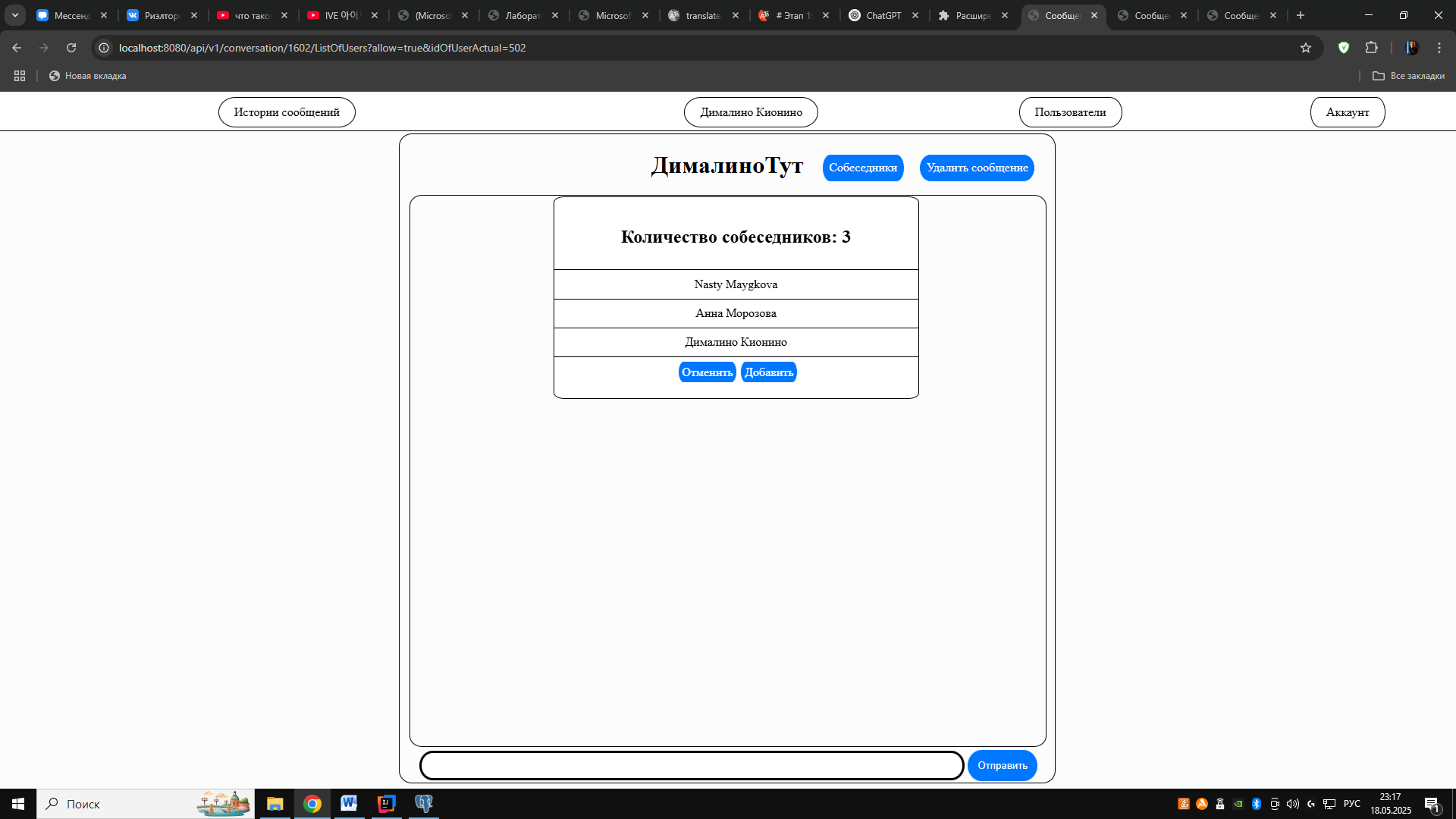


Рисунок 24 – успешно проверил наличие новых пользователей в беседе.

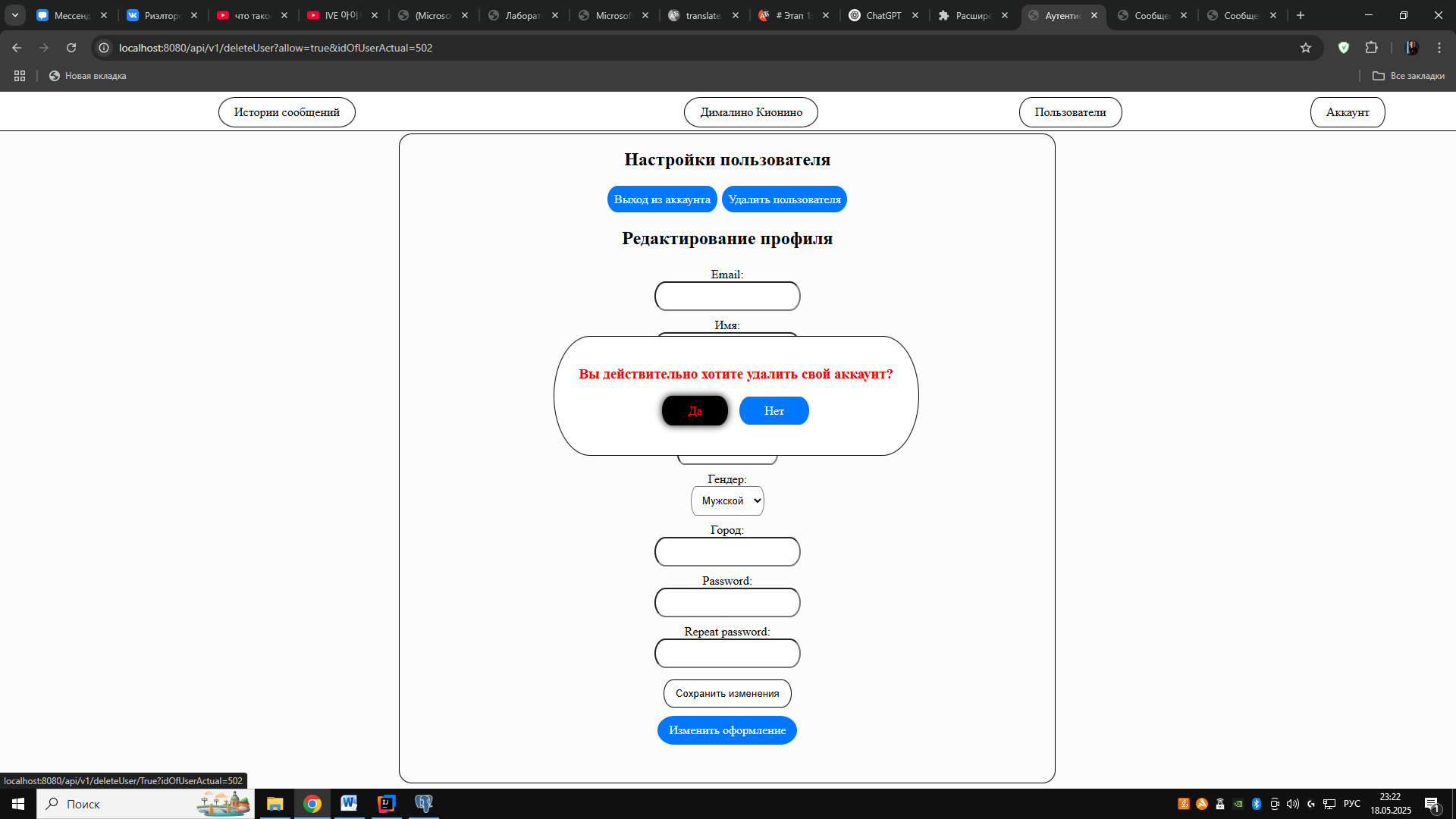


Рисунок 25 – успешно открыл окно с дополнительным подтверждением для удаления аккаунта.

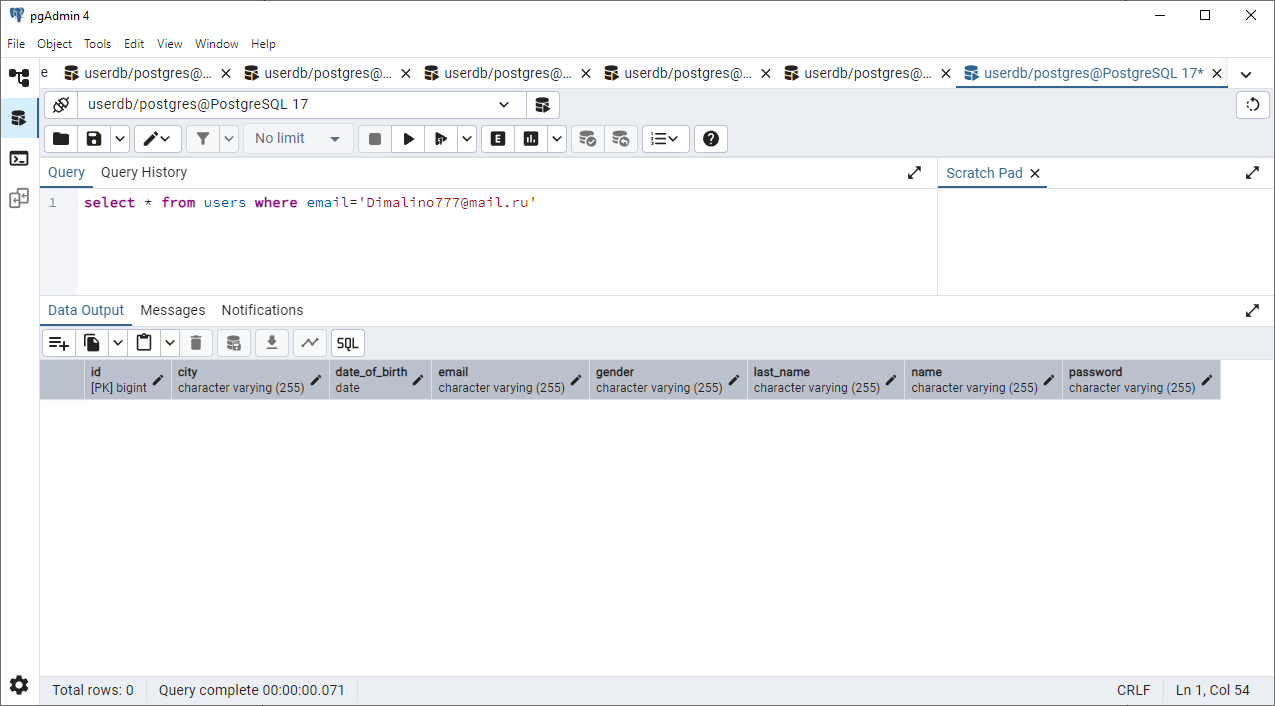


Рисунок 26 – успешно проверил уничтожение аккаунта в базе данных.

Таблица 12 — 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.

**5** **Руководство по программному продукту**

Исходный код данного приложения доступен для общественности, что позволяет любому желающему вносить изменения или улучшения.

Программа разработана для использования под операционной системой MS Windows и Linux и может быть запущена на любом компьютере, где установлены данные ОС.

**5.1 Руководство программиста**

Программа имеет открытый исходный код и ее проект включает в себя следующие файлы: SseController.java и UserController.java – классы, описывающий работу с клиентом. Conversation.java, Message.java, MidConversation.java, User.java – модели для базы данных. ConversationRepository.java, DataInitializer.java, InMemoryUserDAO.java, MessageRepository.java, MidConversationRepository.java, UserRepository.java – классы для работы с базой данных. UserService.java – интерфейс для работы с сервисами. MemoryUserServiceImpl.java, UserServiceImpl.java – сервисы для реализации бизнес логики. SimpleChatNewJava17Application.java – основной класс для запуска проекта. SimpleChatNewJava17ApplicationTests.java – файл с unit-тестами. README.md – руководство для запуска проекта. scriptForConf.sh – скрипт для создания базы данных и изменения конфигурационного файла.

**5.2 Руководство администратора**

В приложении не предусмотрены специальные настройки для администратора. Приложение можно установить на компьютеры с 64 разрядной ОС. Версия ОС – Windows 10 или выше.

Минимальные технические требования к компьютеру:

1. Процессор с тактовой частотой 1,6 ГГц;
2. ОЗУ 8192 Мб;
3. Клавиатура, мышь

**5.3 Руководство пользователя**

Запуск проекта

В первую очередь нужно подключиться к базе данных и для это нужно запустить скрипт scriptForConf.sh, но сперва нужно дать ему права на выполнение, командой: chmod +x scriptForConf.sh.

После чего нужно будет запустить скрипт командой: ./scriptForConf.sh и указать логин для вашего сервера SQL и пароль, после чего будет создана таблица userdb и будет изменена конфигурация для подключения в application.yaml

Дальше нужно будет собрать проект Maven. Для сборки проекта Maven выполните одну из следующих команд: mvnd clean package или mvn clean package.

После чего, для запуска самого проекта - выполните команду:

java -jar target/SimpleChatNewJava17-0.0.1-SNAPSHOT.jar

После этого откройте браузер и перейдите по адресу: (http://localhost:8080/api/v1/entrance).

Чтобы завершить работу проекта, в командной строке зажмите комбинацию клавиш Ctrl+C.

Больше информации вы найдёте в файле README.md

**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.

**Список литературы**

1. Блох, Джошуа. Эффективное программирование на Java. Москва: ДМК пресс 2020 456 с.
2. Еккаль, Брюс. Философия Java Санкт-Петербург: Питер, 2019, 1162 с.
3. Хоретман, Кей С. Биг Java: ранние объекты Москва – Питер, 2020 1200 с.