МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Веб-приложения для проведения видеоконференций

Выполнил студент Драч Кирилл Алексеевич

(Ф.И.О.)

Руководитель проекта ассистент Кантарович В.С

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Современные технологии разработки web-приложений"

|  |
| --- |
| Специальность: 1-40 05 01 03 «Информационные системы и технологии»  Группа:  1  Студент: Драч К. А. |
| **Тема: Веб-приложения для проведения видеоконференций** | |

**1. Срок сдачи студентом законченной работы**: "16 декабря 2024 г."

**2. Исходные данные к проекту:**

**2.1.** Функционально приложение должно:

* обеспечивать реализацию 2 ролей: гость, зарегистрированный пользователь;
* Обеспечивать функционал для гостя: подключение к видеоконференции с помощью пригласительной ссылки, подключение к видеоконференции по идентификационному коду, использование чата для обмена сообщениями с участниками, участие в видеоконференции с передачей видео и звука, получение аудио- и видеопотока от других участников видеоконференции, возможность регистрации в веб-приложении.
* Обеспечивать функционал для зарегистрированного пользователя: авторизация, изменение данных своего профиля, создание видеоконференций, создание пригласительной ссылки для подключения к видеоконференции, добавление других пользователей в «контакты» и их удаление из «контактов», обмен сообщениями с пользователями из списка «контактов».

**2.2.** Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных, реализовано под разными платформами. Программное средство должно представлять собой web-приложение с асинхронным UI. Отображение, бизнес-логика и хранилище данных должны быть максимально независимы друг от друга для возможности расширения. Развёртывание конечного приложения для последующего использования должно осуществляться с использованием Docker. Диаграммы вариантов использования, классов реализации задачи, последовательности разработать на основе UML. Язык разработки проекта Java, JavaScript, платформа Vaadin, протокол WebRTC. Web-приложение должно быть логически завершенным.

**3. Содержание расчетно-пояснительной записки**

* Введение
* Постановка задачи
* Проектирование веб-приложения
* Разработка веб-приложения
* Тестирование веб-приложения
* Руководство пользователя
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления выполненной курсовой работы:**

* + теоретическая часть курсового проекта должны быть представлены в формате MS Word. Оформление записки должно быть согласно выданным правилам;
  + необходимые схемы, диаграммы и рисунки допускается делать в MS Office Visio, Rational Rose, WS или копии экрана (интерфейс);
  + листинги программы представляются частично в приложении;
  + к записке необходимо приложить CD (DVD), который должен содержать: пояснительную записку, листинги и инсталляцию проекта.

**Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 08.10.2024 |  |
| 2 | Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования | 15.10.2024 |  |
| 3 | Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных) | 22.10.2024 |  |
| 4 | Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса | 22.10.2024 |  |
| 5 | Кодирование программного средства | 19.11.2024 |  |
| 6 | Тестирования и отладка программного средства | 26.11.2024 |  |
| 7 | Оформление пояснительной записки | 03.12.2024 |  |
| 8 | Сдача проекта | 16.12.2024 |  |

**5. Дата выдачи задания 16.09.2024**

Руководитель**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** *В.С. Кантарович*

(подпись)

Задание принял к исполнению **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Содержание**

[Введение 6](#_Toc187827935)

[1 Постановка задачи и обзор аналогичных решений 7](#_Toc187827936)

[1.1 Обзор аналогов 7](#_Toc187827937)

[1.2 Постановка задач 11](#_Toc187827938)

[1.3 Вывод по разделу 11](#_Toc187827939)

[2 Проектирование приложения 12](#_Toc187827940)

[2.1 Выбор средств реализации 12](#_Toc187827941)

[2.2 Описание работы программы 15](#_Toc187827942)

[2.3 Проектирование базы данных 17](#_Toc187827943)

[2.3.1 Логическая схема базы данных 17](#_Toc187827944)

[2.3.2 Описание базы данных 17](#_Toc187827945)

[2.4 Контейнеризация 19](#_Toc187827946)

[2.5 Проектирование основных алгоритмов 19](#_Toc187827947)

[2.6 Вывод по разделу 22](#_Toc187827948)

[3 Разработка приложения 23](#_Toc187827949)

[3.1 Разработка серверной части 23](#_Toc187827950)

[3.1.1 Cтруктура серверной части приложения 23](#_Toc187827951)

[3.1.2 Пример директории *entity* 24](#_Toc187827952)

[3.1.3 Пример директории *service* 25](#_Toc187827953)

[3.2 Разработка клиентской части 26](#_Toc187827954)

[3.3 Контейнеризация приложения с использованием *Docker* 28](#_Toc187827955)

[3.4 Вывод по разделу 28](#_Toc187827956)

[4 Тестирование приложения 30](#_Toc187827957)

[4.1 Валидация форм ввода 30](#_Toc187827958)

[4.2 Проверка валидности ID комнаты видеоконференции 31](#_Toc187827959)

[4.3 Поиск несуществующих пользователей 32](#_Toc187827960)

[4.4 Обработка сценария попытки создания комнаты гостем 32](#_Toc187827961)

[4.5 Вывод по разделу 33](#_Toc187827962)

[5 Руководство пользователя 34](#_Toc187827963)

[5.1 Руководство пользователя 34](#_Toc187827964)

[5.1.1 Регистрация пользователя 34](#_Toc187827965)

[5.1.2 Авторизация пользователя 35](#_Toc187827966)

[5.1.3 Главная страница приложения 35](#_Toc187827967)

[5.1.4 Подключение к видеоконференции 36](#_Toc187827968)

[5.1.5 Страница чата с пользователями 37](#_Toc187827969)

[5.1.6 Страница установления контактов 37](#_Toc187827970)

[5.1.7 Комната видео конференции 38](#_Toc187827971)

[5.2 Руководство по развёртыванию приложения 39](#_Toc187827972)

[5.2.1 Подготовка к развёртыванию 39](#_Toc187827973)

[5.2.2 Структура развёртывания 39](#_Toc187827974)

[5.2.3 Запуск приложения 39](#_Toc187827975)

[5.3 Вывод по разделу 40](#_Toc187827976)

[Заключение 41](#_Toc187827977)

[Список используемых источников 42](#_Toc187827978)

[ПРИЛОЖЕНИЕ А 43](#_Toc187827979)

[ПРИЛОЖЕНИЕ Б 44](#_Toc187827980)

[ПРИЛОЖЕНИЕ В 45](#_Toc187827981)

[ПРИЛОЖЕНИЕ Г 47](#_Toc187827982)

[ПРИЛОЖЕНИЕ Д 50](#_Toc187827983)

[ПРИЛОЖЕНИЕ Е 55](#_Toc187827984)

Введение

В современных организациях важную роль играет эффективное управление средствами коммуникации, особенно в условиях гибридных и удалённых форматов работы. Рост числа сотрудников, работающих в различных режимах, требует внедрения удобных инструментов для проведения видеоконференций, управления участниками и обеспечения беспрепятственной коммуникации.

Целью данного курсового проекта является создание веб-приложения, которое позволит пользователям подключаться к видеоконференциям в роли гостя или зарегистрированного пользователя, используя пригласительные ссылки или идентификационные коды, участвовать в чатах, а также управлять своими профилями и контактами. Приложение будет включать функционал для двух ролей пользователей: гость и зарегистрированный пользователь. Архитектура системы будет разработана с учётом масштабируемости и безопасности, обеспечивая независимость между клиентской частью, серверной логикой и хранилищем данных.

Для реализации проекта выбраны современные веб-технологии, включая Vaadin[1] для создания интуитивно понятного и отзывчивого пользовательского интерфейса, Java[2] и JavaScript[3] для разработки серверной и клиентской логики, а также протокол WebRTC[4] для обработки реального времени аудио и видео коммуникации. Веб-приложение будет контейнеризировано с использованием Docker[5], что обеспечит простоту развертывания и гибкость в управлении инфраструктурой. Для надёжного хранения информации о пользователях, видеоконференциях и контактах будет использоваться база данных PostgreSQL[6].

Актуальность проекта обусловлена растущей необходимостью в эффективных системах видеоконференций, особенно в условиях увеличения числа гибридных и удалённых сотрудников. Создание специализированного веб-приложения для видеоконференций отвечает современным требованиям организаций, предоставляя пользователям возможность легко подключаться и участвовать в видеоконференциях, использовать чат для общения и эффективно управлять своими профилями и контактами. Кроме того, использование протокола WebRTC[4] обеспечивает высококачественную передачу аудио и видео в реальном времени, а Docker[5] облегчает масштабируемое и управляемое развёртывание приложения. Интеграция Vaadin[1] гарантирует интуитивно понятный и удобный интерфейс, улучшая общий пользовательский опыт. Применение PostgreSQL для хранения данных обеспечивает надёжность и устойчивость системы, позволяя эффективно обрабатывать большие объёмы информации и поддерживать стабильную работу приложения.

1 Постановка задачи и обзор аналогичных решений

В ходе исследования аналогичных веб-приложений для проведения видеоконференций были проанализированы несколько платформ с похожими функциональными возможностями. Анализ показал, что большинство этих решений предоставляет базовые функции для организации видеосвязи, такие как передача аудио- и видеопотоков, однако часто не охватывает расширенные потребности пользователей, включая гибкое управление контактами, возможности создания пригласительных ссылок и расширенный функционал для различных ролей пользователей.

Также были изучены статьи и практические примеры, связанные с разработкой систем видеоконференций, что помогло сформировать основу для реализации ключевых функций проекта и углубить понимание подходов к проектированию пользовательского интерфейса и архитектуры приложения. Результаты анализа легли в основу требований и проектирования веб-приложения для организации видеоконференций, ориентированного на современный подход к удалённой коммуникации.

1.1 Обзор аналогов

*Jitsi Meet*[6] — это бесплатная и открытая платформа для видеоконференций, разработанная сообществом. Она предоставляет возможности для проведения видеозвонков без необходимости регистрации или установки дополнительного программного обеспечения.

Основные особенности *Jitsi Meet* включают:

* Интуитивно понятный интерфейс: простая навигация и удобный интерфейс.
* **Безопасность:** поддержка сквозного шифрования для обеспечения конфиденциальности коммуникаций.
* Интеграция с другими сервисами: возможность интеграции с различными инструментами и платформами через API.
* Поддержка экранного совместного использования: функция демонстрации экрана позволяет участникам делиться своим рабочим столом.
* Мобильные приложения: доступность на устройствах под управлением iOS и Android обеспечивает гибкость использования.
* Недостатки *Jitsi Meet:*
* Ограниченные возможности аналитики: встроенные инструменты аналитики менее развиты по сравнению с коммерческими решениями.
* Требования к серверным ресурсам: для самостоятельного хостинга требуется обеспечение серверной инфраструктуры и ресурсов.
* Сложность настройки для крупных организаций: внедрение и масштабирование могут потребовать значительных технических знаний и усилий.

Главная страница и страница подключения к комнате представлены на рисунке 1.1.

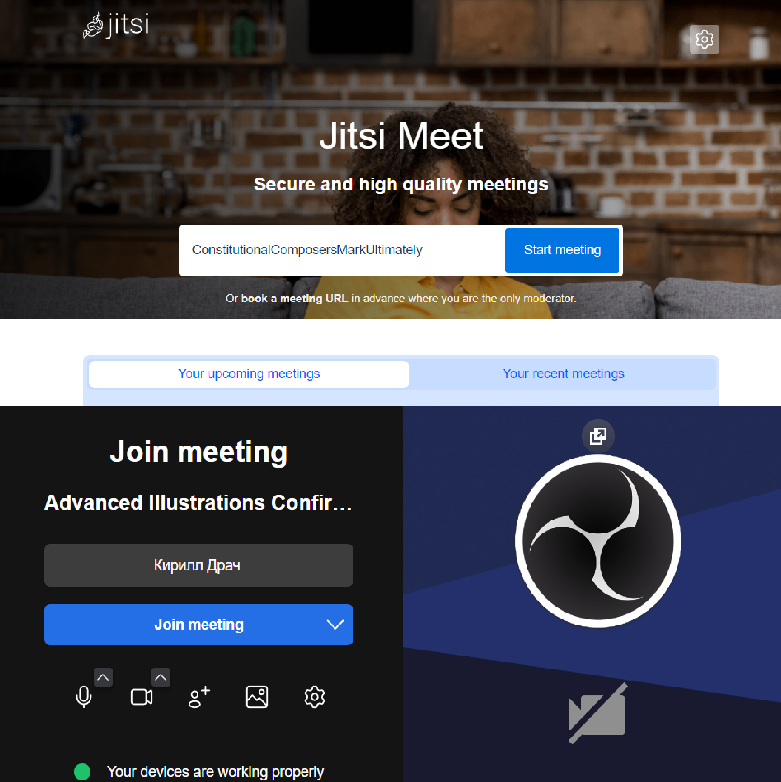


Рисунок 1.1 – Главная страница и страница подключения к комнате «*Jitsi Meet*»

*Amazon Chime*[7] — это сервис для проведения онлайн-встреч, видеоконференций, звонков и чат-коммуникаций, разработанный компанией *Amazon Web Services (AWS)*. Он предназначен для корпоративных клиентов и интегрируется с другими сервисами *AWS*. Основные особенности *Amazon Chime* включают:

* + Интеграция с *AWS*: глубокая интеграция с другими сервисами *AWS*, что позволяет использовать дополнительные инструменты для аналитики и управления. Системы безопасности: интеграция с системами контроля доступа для обеспечения безопасности бронированных ресурсов.
  + Совместное использование экрана и документов: функции для демонстрации экрана, обмена файлами и совместной работы над документами.
  + Запись встреч: возможность записи видеоконференций для последующего просмотра и анализа.
  + Мобильные и настольные приложения: доступность на различных устройствах для гибкости использования.

Недостатки *Amazon Chime*:

* Стоимость: высокие цены могут быть непривлекательными для небольших и средних организаций.
* Ограниченные функции в сравнении с некоторыми конкурентами: некоторые продвинутые функции могут отсутствовать или быть менее развитыми по сравнению с другими платформами.

Главная страница представлена на рисунке 1.2.

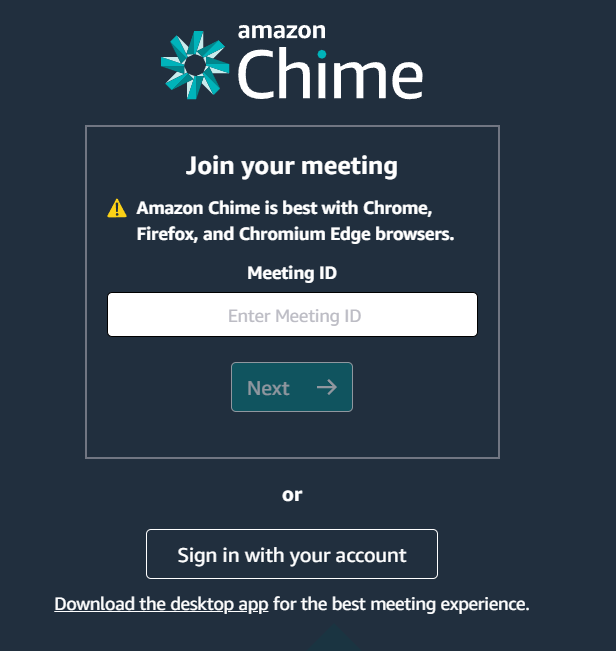


Рисунок 1.2 – Главная страница «*Amazon Chime*»

*Zoho Meeting*[8] — это облачная платформа для проведения вебинаров и видеоконференций, разработанная компанией *Zoho* *Corporation*. Платформа предназначена для малых и средних предприятий, предлагая инструменты для эффективной онлайн-коммуникации и сотрудничества.

Главная страница представлена на рисунке 1.3.

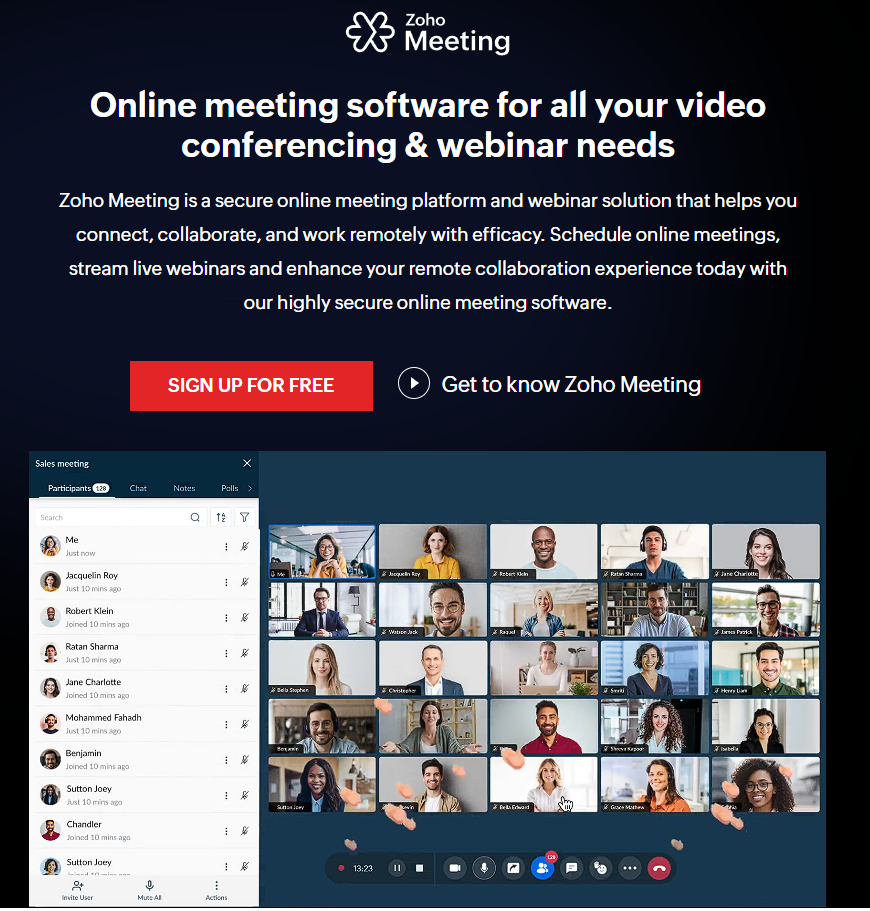


Рисунок 1.3 – Главная страница «*Zoho Meeting*»

Основные особенности *Zoho Meeting* включают:

* + Видеоконференции: возможность проведения видеозвонков с поддержкой HD-качественного видео и аудио, обеспечивая четкую связь между участниками.
  + Совместное использование экрана и документов: возможность демонстрации экрана, обмена файлами и совместного редактирования документов в режиме реального времени.
  + Чат и взаимодействие: встроенные инструменты для обмена сообщениями, проведения опросов и голосований, что способствует интерактивному взаимодействию участников.
  + Поддержка мобильных устройств: доступность через мобильные приложения обеспечивает удобство использования для сотрудников на ходу.
  + Интеграция с другими сервисами *Zoho*: глубокая интеграция с другими продуктами *Zoho,* такими как *Zoho CRM, Zoho Projects* и *Zoho Desk*, для упрощения управления бизнес-процессами.

Недостатки *Zoho Meeting*:

* + Ограниченная функциональность для малых организаций: некоторые функции могут быть избыточными для небольших команд.
  + Необходимость обучения: пользователям может потребоваться время для освоения всех возможностей платформы.
  + Стоимость: подписка может быть дорогой для небольших организаций с ограниченным бюджетом.

1.2 Постановка задач

На основе проведённого анализа и требований к разработке веб-приложения для организации видеоконференций были сформулированы следующие задачи:

* + Разработка системы регистрации и авторизации: создать безопасный функционал для регистрации и авторизации пользователей, обеспечивая защиту и безопасное хранение учётных данных.
  + Реализация системы ролей: внедрить поддержку двух ролей — гость и зарегистрированный пользователь. Каждая роль должна обладать уникальными возможностями для эффективного взаимодействия с приложением. Гости смогут подключаться к видеоконференциям без регистрации, в то время как зарегистрированные пользователи получат расширенные функции управления своими профилями и контактами.
  + Функционал для гостей: функционал для гостей включает подключение к видеоконференциям через пригласительные ссылки или идентификационные коды, участие в чате, передачу и получение аудио- и видеопотоков, а также возможность регистрации для перехода в статус зарегистрированного пользователя.
  + Функционал для зарегистрированных пользователей: предоставить управление профилем, создание видеоконференций с генерацией пригласительных ссылок, добавление и удаление контактов, а также обмен сообщениями с ними.
  + Проектирование и разработка базы данных: создать эффективную и масштабируемую структуру базы данных PostgreSQL, обеспечивающую хранение информации о пользователях, контактных списках и сообщениях.
  + Контейнеризация приложения: использовать *Docker* для контейнеризации серверной части и базы данных, обеспечивая изоляцию и упрощение процесса развертывания приложения на различных платформах.
  + Тестирование и обеспечение безопасности: провести всестороннее тестирование всех компонентов приложения и внедрить меры по обеспечению безопасности данных пользователей и устойчивости системы к потенциальным угрозам.

Выполнение перечисленных задач позволит создать функциональное, безопасное и удобное веб-приложение для организации видеоконференций, удовлетворяющее современные требования к удалённой коммуникации и способствующее улучшению качества взаимодействия между участниками.

1.3 Вывод по разделу

Анализ платформ для видеоконференций показал, что успешное приложение должно обеспечивать интуитивно понятный интерфейс, удобное подключение участников и качественную передачу аудио- и видеопотоков. Важно поддерживать роли гостя и зарегистрированного пользователя с расширенным функционалом. Использование WebRTC и Docker повысит производительность, масштабируемость и удобство развертывания.

2 Проектирование приложения

Проект, разработанный в рамках данной работы, представляет собой веб-приложение с клиент-серверной архитектурой, предназначенное для проведения видеоконференций и эффективного управления процессом взаимодействия между участниками. Архитектура приложения включает две основные части: клиентскую и серверную.

Клиентская часть обеспечивает взаимодействие с пользователями двух ролей — гостем и зарегистрированным пользователем. Гость может подключаться к видеоконференциям с помощью пригласительных ссылок или идентификационных кодов, использовать чат для обмена сообщениями и участвовать в передаче аудио- и видеопотоков. Зарегистрированный пользователь, помимо вышеуказанных возможностей, может создавать собственные видеоконференции, управлять списком контактов и обмениваться личными сообщениями с добавленными контактами.

Серверная часть отвечает за обработку запросов, управление ролями и правами доступа, а также взаимодействие с базой данных для хранения информации о пользователях, конференциях, контактных данных и сообщениях. Применение протокола *WebRTC* обеспечивает качественную передачу аудио- и видеопотоков в реальном времени, а использование *Docker* позволяет упростить процесс развертывания и масштабирования приложения на различных платформах.

Такое разделение функциональности и использование современных технологий способствует повышению производительности и надёжности приложения, обеспечивая его безопасность, масштабируемость и удобство использования для всех категорий пользователей.

2.1 Выбор средств реализации

При разработке веб-приложения для проведения видеоконференций был проведён тщательный анализ доступных технологий и инструментов, позволяющих эффективно реализовать все заявленные функциональные требования. Основными критериями выбора стали производительность, удобство разработки, масштабируемость, безопасность и возможность кроссплатформенной интеграции.

Для клиентской части приложения был выбран фреймворк ***Vaadin***, обеспечивающий создание интуитивно понятного и отзывчивого пользовательского интерфейса на базе *Java*, а также возможность разработки *UI* и серверной логики в едином технологическом стеке. Использование *Vaadin* ускоряет процесс разработки за счёт встроенных компонентов интерфейса и поддерживает асинхронные запросы для повышения интерактивности веб-приложения.

Серверная часть будет реализовываться с использованием ***Java***, что обеспечивает высокую производительность, безопасность и гибкость для реализации бизнес-логики и обработки данных. *Java* была выбрана благодаря её стабильности, поддержке многопоточности и широкому сообществу разработчиков.

В качестве системы управления базами данных используется *PostgreSQL*, известная своей надёжностью, поддержкой сложных *SQL*-запросов и совместимостью с *Java*. *PostgreSQL* позволяет эффективно управлять данными о пользователях, видеоконференциях, контактных списках и сообщениях, обеспечивая стабильную работу приложения даже при высокой нагрузке.

Для реализации функций видеоконференций и передачи аудио- и видеопотоков в реальном времени используется протокол *WebRTC*, позволяющий организовать прямую связь между клиентами без промежуточных серверов.

Язык *JavaScript* применяется для поддержки клиентской интерактивности, особенно в части динамических элементов интерфейса и обработки данных в реальном времени, взаимодействуя с *Vaadin*-компонентами и *WebRTC*.

Таким образом, выбранный стек технологий — *Vaadin*, *Java*, *PostgreSQL*, *JavaScript* и *WebRTC* — обеспечивает высокую производительность, масштабируемость, безопасность и удобный пользовательский опыт, что особенно важно для разработки современного веб-приложения для проведения видеоконференций.

2.1.1 Клиентская часть

При разработке веб-приложения для проведения видеоконференций был проведён анализ технологий, позволяющих эффективно реализовать заявленные функциональные требования с учётом производительности, безопасности, масштабируемости и удобства разработки. В результате для реализации клиентской части был выбран следующий стек:

* + *Vaadin*: используется для создания интуитивно понятного и отзывчивого пользовательского интерфейса. *Vaadin* обеспечивает компонентную архитектуру, упрощая разработку *UI* с использованием *Java* без необходимости глубокого взаимодействия с фронтенд-фреймворками.
  + *Java*: применяется для серверной части, управления бизнес-логикой и обработки данных. Язык обеспечивает стабильность, производительность и безопасность, поддерживая масштабируемость системы.
  + *PostgreSQL*: используется для надёжного хранения данных о пользователях, конференциях, сообщениях и контактных данных. База данных обеспечивает высокую производительность, надёжность и поддержку сложных *SQL*-запросов.
  + *JavaScript* (*WebRTC*): ирименяется для реализации видеоконференций в реальном времени с поддержкой передачи аудио- и видеопотоков между участниками без промежуточных серверов. *WebRTC* интегрирован с компонентами Vaadin для динамического отображения медиапотоков.
  + *Docker*: используется для контейнеризации серверной части и базы данных, обеспечивая изоляцию и упрощение процесса развертывания приложения на различных платформах. *Docker* способствует стабильности окружения и облегчает масштабирование приложения.

Выбранный стек технологий — *Vaadin*, *Java*, *PostgreSQL*, *JavaScript* и *Docker* — обеспечивает высокую производительность, безопасность и удобство использования, позволяя создать современное и масштабируемое веб-приложение для организации видеоконференций.

2.1.2 Серверная часть

Серверная часть веб-приложения для видеоконференций будет реализована на Java с использованием фреймворка Vaadin, что позволяет объединить серверную и клиентскую логику в единой среде разработки.

Основными факторами выбора стали:

* + Надёжность и зрелость: *Java* и *Vaadin* проверены временем, поддерживают многопоточность, масштабируемость и стабильность серверных решений.
  + Быстрая разработка: *Vaadin* предоставляет готовые UI-компоненты и возможности для упрощённого взаимодействия с серверной логикой, что сокращает время разработки.Интеграция с *PostgreSQL*: Django легко интегрируется с базой данных *PostgreSQL*, обеспечивая эффективное управление данными, поддержку сложных запросов и высокую производительность.
  + Интеграция с *PostgreSQL*: Простая интеграция с базой данных *PostgreSQL* через ORM-решения для безопасного и производительного управления данными.
  + Безопасность: Встроенные механизмы защиты от *SQL*-инъекций, *XSS* и *CSRF*-атак обеспечивают защиту данных пользователей.
  + Контейнеризация с *Docker*: Использование *Docker* для серверной части позволяет создать изолированное и воспроизводимое окружение, упрощая развертывание приложения на различных платформах.

Серверная часть отвечает за обработку запросов, управление ролями пользователей (гость и зарегистрированный пользователь), реализацию бизнес-логики и безопасное хранение данных о пользователях, конференциях и сообщениях. Использование *WebRTC* для передачи медиапотоков в реальном времени и *PostgreSQL* для хранения данных обеспечивает высокую производительность и масштабируемость системы.

2.1.3 Централизованная база данных

В качестве системы управления базами данных в веб-приложении для видеоконференций выбрана *PostgreSQL*. Этот выбор обусловлен рядом ключевых преимуществ:

* Надёжность и безопасность: *PostgreSQL* обеспечивает высокую надёжность данных и поддерживает механизмы безопасности, включая шифрование данных, а также гибкую систему управления правами доступа.
* Гибкость и расширяемость: поддержка различных типов данных и возможность расширения функциональности через плагины позволяют адаптировать базу под нужды приложения.
* Сообщество и поддержка: обширное сообщество разработчиков и подробная документация упрощают сопровождение и настройку базы данных.

База данных PostgreSQL хранит информацию о пользователях, конференциях, контактных списках, сообщениях и ролях, обеспечивая целостность данных с помощью механизмов транзакций и ограничений.

2.1.4 Архитектура приложения

Проект разделён на клиентскую и серверную части, которые взаимодействуют между собой и базой данных. Такой подход обеспечивает удобное масштабирование, безопасность и высокую производительность системы.

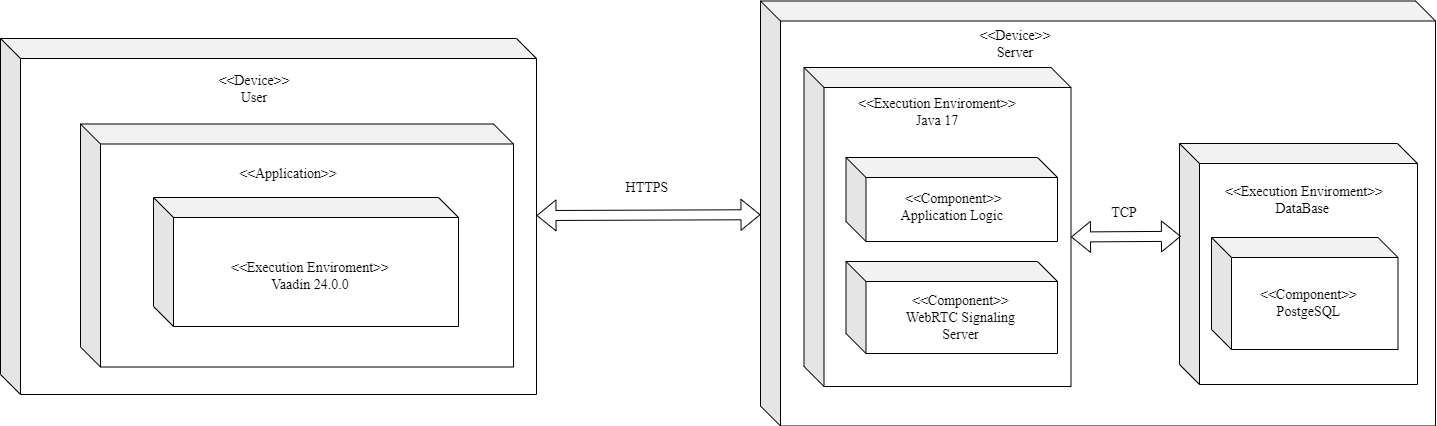


Рисунок 2.1 – Структурная схема приложения

Клиентская часть разработана с использованием фреймворка *Vaadin* на *Java*. *Vaadin* предоставляет мощный компонентный подход для создания интуитивно понятного веб-интерфейса, минимизируя необходимость работы с *JavaScript* напрямую. Клиент отвечает за отображение интерфейса, управление пользовательскими сессиями, подключение к видеоконференциям, управление контактами и профилем пользователя.

Серверная часть будет реализована на *Java* с *Vaadin* *Backend*. Сервер обрабатывает *HTTP*-запросы, управляет ролями пользователей (гость, зарегистрированный пользователь), отвечает за авторизацию, обработку данных о видеоконференциях и контактных данных. Взаимодействие с клиентом осуществляется по протоколу *HTTPS*.

2.2 Описание работы программы

В данном разделе подробно описывается функционирование веб-приложения для проведения видеоконференций, а также представлены основные диаграммы, демонстрирующие архитектуру системы и взаимодействие её компонентов.

2.2.1 Диаграмма вариантов использования

После запуска веб-приложения серверная часть на *Java* с *Vaadin* настраивается для обработки входящих *HTTP*-запросов на установленном порту, что позволяет клиентской части (*Vaadin* *UI*) обращаться к серверу для выполнения различных операций. Приложение поддерживает два типа пользователей — гостя и зарегистрированного пользователя, каждый из которых обладает определённым набором функций, представленных на диаграмме вариантов использования. Основные сценарии использования и функции приложения представлены на *use* *case*-диаграмме, изображённой на рисунке 2.2.

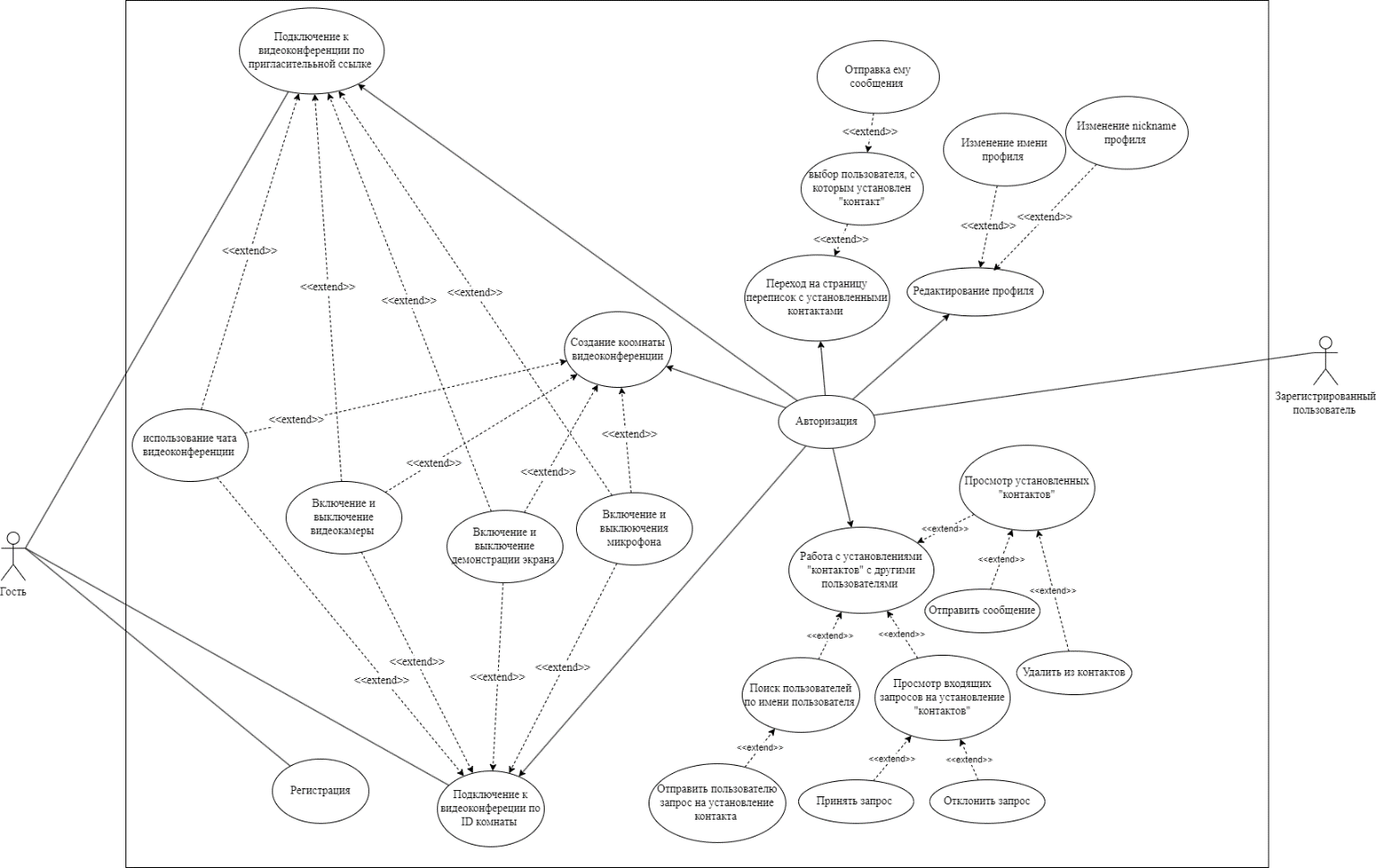


Рисунок 2.2 – Диаграмма вариантов использования

Рисунок 2.2 демонстрирует основные сценарии взаимодействия пользователя с приложением. Подробное описание сценариев:

### Гость:

* **Регистрация**: гость может создать учётную запись, заполнив регистрационную форму, после чего получает статус зарегистрированного пользователя.
* Подключение к конференции: возможность подключения через пригласительную ссылку или с использованием уникального идентификационного кода комнаты.
* Функционал во время конференции: гость может включать и выключать камеру, микрофон, демонстрацию экрана, а также пользоваться встроенным чатом для обмена сообщениями с другими участниками.

### Зарегистрированный пользователь:

* Авторизация: пользователь вводит свои учётные данные для получения доступа к расширенным возможностям приложения.
* Создание и управление конференциями: возможность создания новых конференций с генерацией пригласительных ссылок или ID комнаты.
* Управление профилем: возможность редактировать профиль, изменять имя и никнейм.
* Работа с контактами: добавление пользователей в список контактов, отправка и получение запросов на добавление в контакты, удаление контактов.
* Обмен сообщениями: возможность отправки сообщений пользователям из списка контактов и ведения диалогов через встроенный чат.

2.3 Проектирование базы данных

2.3.1 Логическая схема базы данных

База данных спроектирована для обеспечения всех функциональных требований веб-приложения для проведения видеоконференций, включая управление пользователями, сообщениями и контактами. В качестве системы управления базами данных (СУБД) выбрана *PostgreSQL*. Логическая схема базы данных представлена на рисунке 2.4.

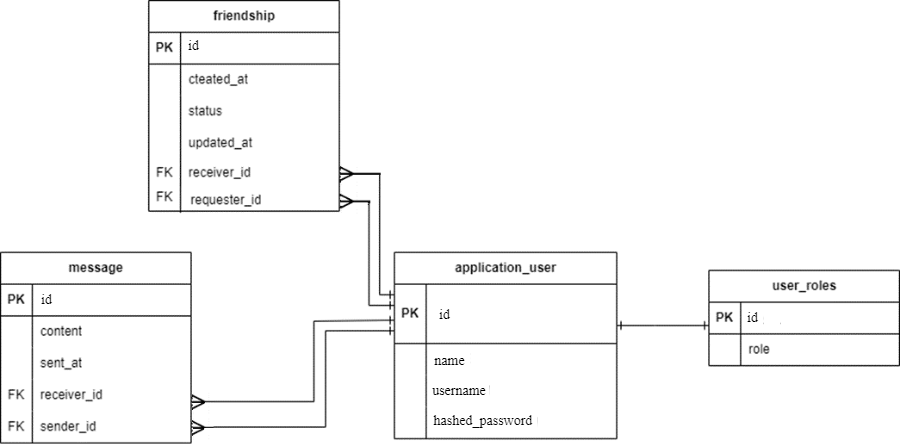


Рисунок 2.4 – Модель базы данных

Такое проектирование позволяет обеспечить целостность данных и эффективное управление пользовательскими взаимодействиями в системе видеоконференций.

2.3.2 Описание базы данных

В таблице «*application\_user*» хранятся все зарегистрированные пользователи. Описание структуры таблицы «*application\_user*» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «*application\_user*»

|  |  |  |
| --- | --- | --- |
| **Название столбца** | **Тип, ограничение целостности** | **Описание столбца** |
| *id* | *int, primary key* | Уникальный идентификатор пользователя. |
| *name* | *varchar(150), not null* | Имя пользователя. |
| *username* | *varchar(150), unique, not null* | Логин пользователя, уникальный. |
| *hashed\_password* | *varchar(256), not null* | Хэшированный пароль для безопасности. |

В таблице «*user\_roles*» хранится информация о ролях пользователей. Описание структуры таблицы «*user\_roles*» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «*user\_roles*»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| *user\_id* | *int, primary key, foreign key* | Ссылка на идентификатор пользователя. |
| *role* | *varchar(50), not null* | Роль пользователя (гость, зарегистрированный пользователь). |

Таблица «*friendship*» хранится информация о запросах на добавление в контакты. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «*friendship*»

|  |  |  |
| --- | --- | --- |
| **Название столбца** | **Тип, ограничение целостности** | **Описание столбца** |
| *id* | *int, primary key* | Уникальный идентификатор запроса. |
| *created\_at* | *timestamp, not null* | Дата создания запроса. |
| *status* | *varchar(50), not null* | Текущий статус запроса. |
| *receiver\_id* | *int, foreign key* | Получатель запроса. |
| *requester\_id* | *int, foreign key* | Отправитель запроса. |

В таблице «*message*» хранится информация о сообщениях между пользователями. Описание структуры таблицы «*message*» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «*message*»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| *id* | *int, primary key* | Уникальный идентификатор сообщения. |
| *content* | *text, not null* | Текст сообщения. |
| *sent\_at* | *timestamp, not null* | Дата и время отправки сообщения. |
| *receiver\_id* | *int, foreign key* | Получатель сообщения. |
| *sender\_id* | *int, foreign key* | Отправитель сообщения. |

Таким образом, при проектировании базы данных используется 4 таблицы, включая связи типа один-к-одному и один-ко-многим. Такая архитектура базы данных обеспечивает целостность данных, гибкость в управлении пользовательской информацией, высокую производительность и безопасность хранения данных.

2.4 Контейнеризация

Контейнеризация веб-приложения для проведения видеоконференций с использованием *Docker* обеспечивает изоляцию и консистентность среды выполнения для основных компонентов системы, включая серверную часть и базу данных. Применение *Docker* позволяет создать отдельные контейнеры для *Java* *Vaadin* *Backend* и базы данных *PostgreSQL*, что исключает конфликты зависимостей и упрощает управление версиями используемых технологий.

Использование *Docker* способствует унификации среды разработки и продакшн-развертывания, минимизируя риски, связанные с различиями в конфигурациях на этапах разработки и эксплуатации. Это позволяет разработчикам быстро развертывать веб-приложение на различных платформах без дополнительных настроек, обеспечивая стабильность и надёжность системы.

Контейнеризация также упрощает масштабирование приложения. В случае увеличения количества участников видеоконференций можно оперативно развернуть дополнительные экземпляры контейнеров серверной части или базы данных для балансировки нагрузки, что повышает гибкость и устойчивость системы к изменениям трафика.

Процесс обновления и поддержки системы также упрощается благодаря контейнеризации. Обновления серверной части или базы данных могут выполняться путём замены контейнеров на обновлённые версии без необходимости ручных изменений конфигурации, что снижает время простоя и повышает доступность сервиса. Для хранения данных база данных *PostgreSQL* использует выделенные тома *Docker*, что обеспечивает надёжное резервное копирование и упрощает восстановление данных.

Таким образом, использование *Docker* в данном проекте позволяет создать управляемую и стабильную инфраструктуру, поддерживающую надёжное взаимодействие между серверной частью и базой данных. Это обеспечивает высокую производительность, безопасность и удобство администрирования системы, что особенно важно для поддержания стабильной работы приложения для видеоконференций.

2.5 Проектирование основных алгоритмов

2.5.1 Алгоритм подключения пользователя по *ID* комнаты

Процесс подключения пользователя по *ID* комнаты в веб-приложении для проведения видеоконференций включает несколько последовательных этапов, обеспечивающих корректное подключение и исключение ошибок ввода.

Изначально пользователь открывает интерфейс веб-приложения и вводит уникальный идентификатор комнаты в соответствующее текстовое поле. Клиентская часть приложения, разработанная с использованием *Vaadin*, принимает введённый *ID* комнаты и формирует *HTTP*-запрос к серверной части.

После получения данных серверная часть выполняет проверку *ID* комнаты на соответствие установленным требованиям

В случае, если *ID* комнаты не соответствует требованиям (например, неправильная длина, некорректные символы или отсутствующая комната в базе данных), сервер возвращает клиенту сообщение об ошибке с указанием причины сбоя: «*ID* не соответствует формату». В этом случае пользователь может повторить попытку ввода данных.

Этот процесс обеспечивает надёжность подключения, исключает ошибки ввода данных и поддерживает безопасное взаимодействие между пользователем и сервером, предотвращая доступ к несуществующим или закрытым конференциям.

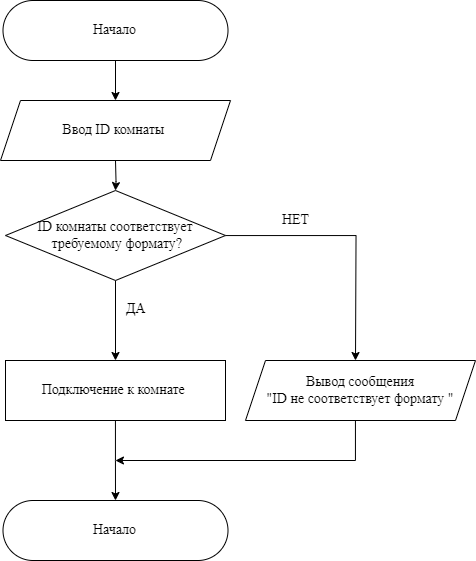


Рисунок 2.5 – Схема алгоритма подключения пользователя по *ID*

Этот процесс обеспечивает надёжность подключения, исключает ошибки ввода данных и поддерживает безопасное взаимодействие между пользователем и сервером, предотвращая доступ к несуществующим или закрытым конференциям.

2.5.2 Алгоритм добавления в контакты

Процесс добавления пользователя в контакты в веб-приложении для проведения видеоконференций включает несколько этапов, направленных на обеспечение корректного поиска и отправки запроса.

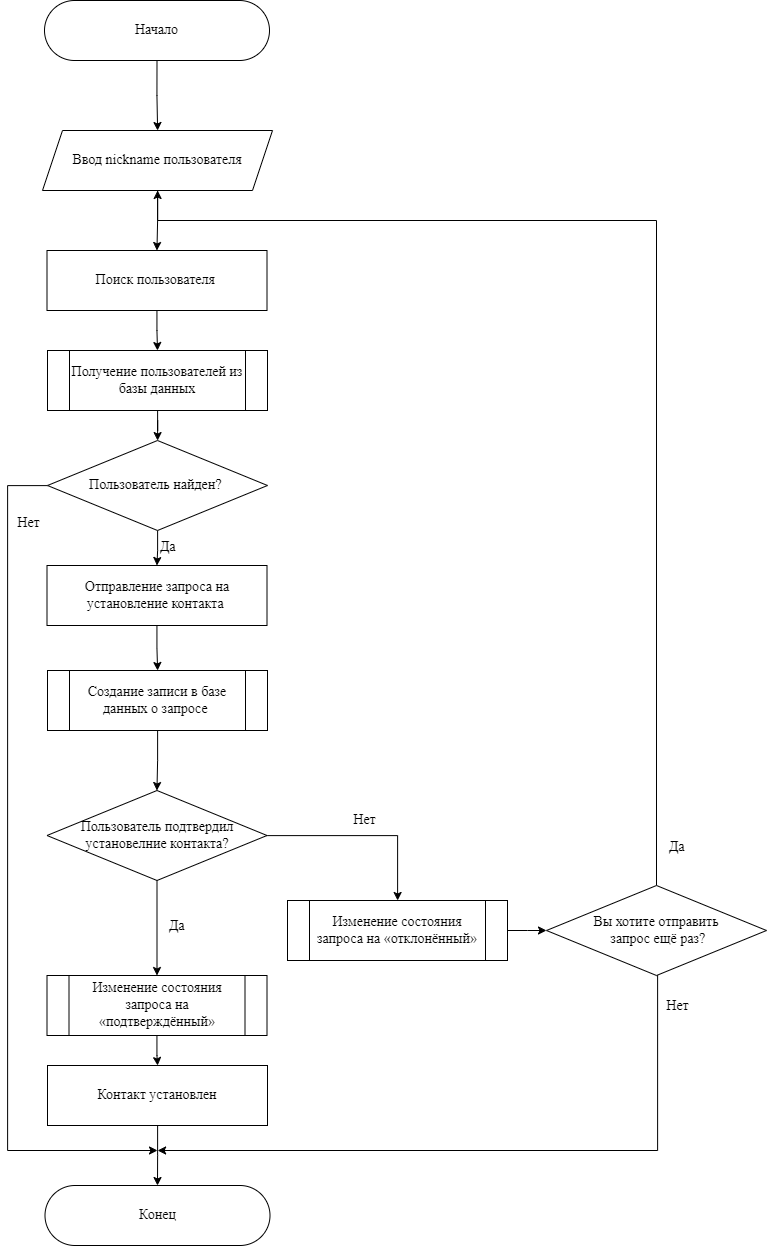


Рисунок 2.6 – Алгоритм добавления в контакты

Данный алгоритм гарантирует, что только согласованные пользователи могут добавляться в контакты, исключая возможность автоматического добавления без подтверждения. Это повышает безопасность взаимодействия в приложении и предотвращает нежелательные запросы.

2.6 Вывод по разделу

Проектируемое веб-приложение для проведения видеоконференций представляет собой решение с чётким разделением клиентской и серверной части. Интерфейс, созданный с использованием *Vaadin*, обеспечивает удобство взаимодействия с пользователем, высокую отзывчивость и совместимость с различными устройствами.

Серверная часть будет реализована на *Java (Vaadin Backend)* и отвечает за обработку данных, управление сессиями и валидацию пользовательских действий. База данных *PostgreSQL* используется для хранения информации о пользователях, контактах и сообщениях, обеспечивая целостность данных и поддержку сложных связей.

Контейнеризация с помощью *Docker* упрощает развертывание и поддержание приложения, позволяя запускать серверную часть и базу данных в изолированных контейнерах, что минимизирует риски конфликтов зависимостей и упрощает обновления.

Алгоритмы подключения пользователя по *ID* комнаты и добавления в контакты проработаны с учётом пользовательского опыта, исключая ошибки ввода и повышая безопасность взаимодействия. В целом, архитектура системы ориентирована на стабильную работу, масштабируемость и возможность расширения функционала в будущем.

3 Разработка приложения

Разработка веб-приложения для проведения видеоконференций была выполнена с использованием современных технологий, обеспечивающих стабильность, производительность и удобство для пользователей. Основной упор сделан на создание интуитивно понятного интерфейса, безопасность данных и возможность масштабирования системы.

Для клиентской части использовался фреймворк *Vaadin*, позволяющий создавать отзывчивый и кроссплатформенный интерфейс. *Vaadin* предоставляет интегрированные компоненты для разработки веб-интерфейсов с минимальными затратами на фронтенд-программирование, что ускоряет процесс разработки и упрощает взаимодействие с серверной частью.

На серверной стороне использовался *Java (Vaadin Backend)*, обеспечивающий обработку запросов, реализацию бизнес-логики и управление пользователями. Для хранения данных о пользователях, контактах и сообщениях была выбрана база данных *PostgreSQL*, взаимодействие с которой осуществляется через встроенные средства *Java* для *ORM* (*Object-Relational Mapping*).

Контейнеризация с помощью *Docker* была использована для упрощения развертывания серверной части и базы данных. Это позволяет запускать систему в изолированных контейнерах, минимизируя конфликты зависимостей и упрощая процесс обновления приложения.

3.1 Разработка серверной части

3.1.1 Cтруктура серверной части приложения

На рисунке 3.1 представлена структура серверной части приложения, включающая в себя директории, конфигурационные файлы и файлы скриптов.

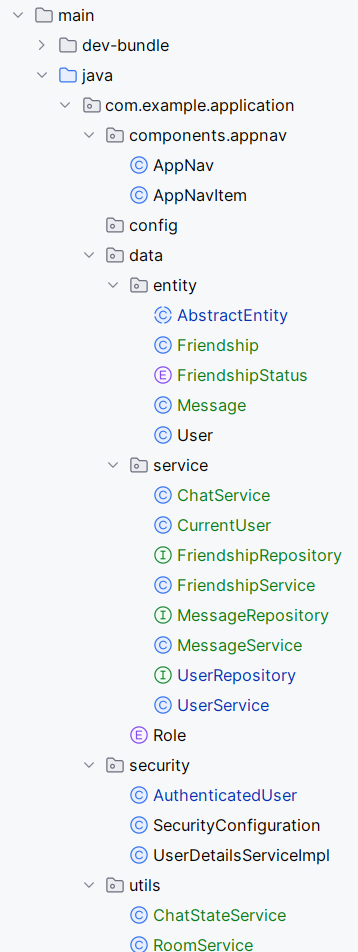
****

Рисунок 3.1 – Структура серверной части

В Серверная часть приложения на *Java* разделена на модули, обеспечивающие взаимодействие с базой данных, бизнес-логику и безопасность. В директории *data* находятся сущности, такие как *User* и *Message*, которые связаны с таблицами базы данных *PostgreSQL*. Сервисы из директории *service* реализуют ключевые алгоритмы, включая управление контактами и отправку сообщений, взаимодействуя с репозиториями. Безопасность обеспечивается в директории *security*, где находятся классы для аутентификации и настройки доступа. Такая структура упрощает масштабирование, поддержку и добавление новых функций.

3.1.2 Пример директории *entity*

Для более детального примера рассмотрим директорию *entity*, представленную на рисунке 3.2.

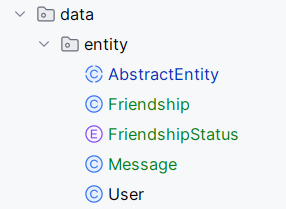
****

Рисунок 3.2 – Содержимое директории «*entity*»

**Директория включает в себя следующее:**

* ***AbstractEntity –* базовый класс, содержащий общие поля, такие как идентификатор (ID) и временные метки (created\_at, updated\_at). Он служит основой для других сущностей, наследующих эти общие свойства, чтобы избежать дублирования кода. Приложение А.**
* ***User –* сущность, представляющая пользователя приложения. Класс содержит основные поля, такие как имя (name), логин (username), пароль (hashed\_password) и связи с другими сущностями, например, Friendship для управления контактами.**
* *Message* – сущность, описывающая сообщения между пользователями. Включает поля *content* (текст сообщения), *sent\_at* (время отправки) и связи с отправителем (*sender\_id*) и получателем (*receiver\_id*).
* *Friendship* – сущность, отвечающая за хранение данных о дружбе между пользователями. Содержит идентификаторы участников (*requester\_id, receiver\_id*) и статус дружбы (*status*), а также временные метки.

Таким образом, директория *entity* группирует все классы, описывающие ключевые объекты базы данных. Она упрощает работу с данными и обеспечивает их структурированное представление для дальнейшего использования в бизнес-логике и контроллерах.

3.1.3 Пример директории *service*

На рисунке 3.3 представлена структура директории *service*, которая играет ключевую роль в реализации бизнес-логики серверной части приложения. Этот каталог объединяет классы и методы, отвечающие за обработку пользовательских запросов, управление контактами, обработку сообщений и другие важные функции. Здесь находятся основные сервисы (*FriendshipService, MessageService, UserService*), которые взаимодействуют с репозиториями (*FriendshipRepository*, *MessageRepository*, *UserRepository*) для выполнения операций с базой данных. Директория служит связующим звеном между слоями данных и контроллерами, обеспечивая логику работы приложения.

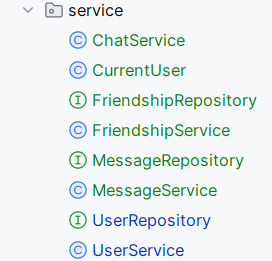


Рисунок 3.3 – Структура приложения *service*

*FriendshipService* – отвечает за обработку запросов, связанных с добавлением пользователей в контакты, подтверждением и отклонением запросов на дружбу. Этот файл реализует основную бизнес-логику взаимодействия между пользователями, включая изменение статуса дружбы и проверку условий для успешного добавления в контакты.

*MessageService* – предоставляет функциональность для отправки, получения и удаления сообщений между пользователями. Он включает обработку временных меток отправки, а также логику валидации сообщений перед сохранением в базе данных. Этот сервис взаимодействует с репозиторием *MessageRepository* для управления данными сообщений. Реализация *MessageService* представлена в Приложении Б.

*UserService* – отвечает за управление данными пользователей, включая создание новых аккаунтов, редактирование профилей и удаление данных. В данном файле также реализован функционал авторизации и аутентификации пользователей через взаимодействие с *UserRepository*. Реализация *UserService* представлена в Приложении В.

3.2 Разработка клиентской части

Разработка клиентской части веб-приложения для проведения видеоконференций была выполнена с использованием фреймворка Vaadin. Этот подход обеспечивает современный и интуитивно понятный пользовательский интерфейс, который активно взаимодействует с серверной частью, отправляя и получая данные через встроенные инструменты для работы с *HTTP*-запросами.

Модульная структура клиентской части представлена на рисунке 3.4 и включает отдельные модули для реализации ключевых функций приложения, таких как управление видеоконференциями, отправка сообщений, управление профилем и контактами. Каждая директория (*chat*, *conference, friends, login, registration, user*) содержит отдельные представления (*views*), что упрощает поддержку и расширение функционала.

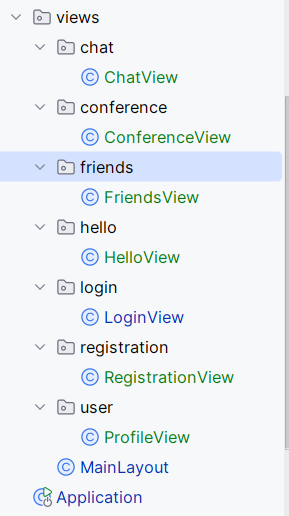


Рисунок 3.4 – Структура клиентской части

В директории *views* каждая страница веб-приложения реализована в виде отдельного представления с использованием *Vaadin*, что позволяет поддерживать удобную навигацию и разделение функциональности.

*ChatView* отвечает за функциональность чата в рамках видеоконференции, предоставляя пользователям возможность отправки и получения сообщений в реальном времени. Представление помечено аннотациями *@PageTitle("Chat")* и *@Route("chat")*, что задает заголовок страницы и маршрут для её отображения. Аннотация *@PermitAll* позволяет доступ к этой странице только для авторизованных пользователей.

*ConferenceView* реализует логику подключения к видеоконференции и управления видеопотоками, включая возможность включения и выключения камеры и микрофона. Для корректного отображения и маршрутизации используется *@PageTitle("Conference")* и *@Route("conference").* Аннотации к классу представлены в листинге 3.1.

|  |
| --- |
| ...  @PageTitle("Conference Room")  @Route(value = "external", layout = MainLayout.class)  @PermitAll  @AnonymousAllowed  public class ConferenceView extends HorizontalLayout implements HasUrlParameter<String> {...} ... |

Листинг 3.1 – Аннотации класса *ConferenceView*

*FriendsView* предоставляет интерфейс для управления контактами, включая добавление, удаление и отправку запросов на установление контакта. Это представление также ограничено аннотацией *@PermitAll*, с маршрутом *@Route("friends")* и заголовком страницы, заданным через @*PageTitle("Friends")*.

|  |
| --- |
| ...  @Autowired  public FriendsView(UserService userService, FriendshipService friendshipService, CurrentUser currentUser) {  this.userService = userService;  this.friendshipService = friendshipService;  this.currentUser = currentUser;  setAlignItems(Alignment.CENTER);  setJustifyContentMode(JustifyContentMode.CENTER);  setSpacing(true);  searchField.setPlaceholder("Enter nickname...");  searchField.setWidth("300px");  Button searchButton = new Button("Search", event -> searchUsers());  HorizontalLayout searchLayout = new HorizontalLayout(searchField, searchButton);  configureUserGrid();  configureFriendsGrid();  add(searchLayout, userGrid, friendsGrid);  loadFriends();  }  ... |

Листинг 3.2 – Конструктор класса *FriendsView*

*LoginView* отвечает за страницу авторизации пользователей. Для её маршрутизации применяется *@Route("login")*, а аннотация *@AnonymousAllowed* указывает, что эта страница доступна без предварительной авторизации. Аналогичный подход используется и в *RegistrationView*, где пользователи могут зарегистрировать новый аккаунт, введя необходимые данные, с доступом, разрешенным для всех через @*AnonymousAllowed*.

*ProfileView* реализует отображение и редактирование данных профиля пользователя. Используются аннотации @*PageTitle*("*Profile*"), @*Route*("*profile*") и @*PermitAll* для ограничения доступа авторизованными пользователями и задания маршрута.

Каждое представление использует механизм маршрутизации *Vaadin*, где в качестве общего контейнера для всех страниц используется класс *MainLayout*, задающий основной макет интерфейса и элементы навигации. В приложении Г продемонстрирован класса *MainLayout*. Такая структура обеспечивает интуитивно понятную навигацию, защищённый доступ к основным ресурсам и логическое разделение компонентов приложения.

3.3 Контейнеризация приложения с использованием *Docker*

Для надёжного и удобного развертывания веб-приложения для видеоконференций была применена контейнеризация с использованием *Docker*. Эта технология позволяет изолировать компоненты приложения, упрощая управление зависимостями и минимизируя риск конфликтов при развёртывании.

В проекте используются два контейнера: контейнер с базой данных *PostgreSQL* и контейнер серверной части на Java. Контейнер базы данных отвечает за хранение пользовательских данных и информации о видеоконференциях. Для сохранности данных при перезапуске контейнера используется том, что позволяет сохранять состояние базы. Серверный контейнер содержит приложение, реализованное на *Vaadin*, обрабатывающее запросы пользователей, управление сессиями и аутентификацию. Взаимодействие между сервером и базой данных осуществляется через переменные окружения, определённые в файле *.env.*

Для управления контейнерами используется *Docker Compose*, который объединяет сервисы в единую сеть и упрощает их настройку. В конфигурации *docker-compose.yml* описаны оба контейнера, их зависимости и сетевые настройки. Такая архитектура позволяет разворачивать приложение одной командой *docker-compose up*. Приложение Д.

Контейнеризация в данном проекте обеспечивает изоляцию компонентов, гибкость обновлений за счёт перезапуска отдельных контейнеров и сохранность данных благодаря подключаемым томам. Использование *Docker* упрощает процесс развертывания и тестирования, делая приложение масштабируемым и легко поддерживаемым.

3.4 Вывод по разделу

В данном разделе была подробно описана разработка веб-приложения для проведения видеоконференций с использованием современных технологий и подходов. Клиентская часть приложения, разработанная на основе *Vaadin*, обеспечила удобный и интуитивно понятный пользовательский интерфейс с чёткой структурой компонентов и маршрутов. Такой подход способствует лёгкому расширению и поддержке функциональности, а использование встроенных инструментов маршрутизации и аннотаций упростило контроль за доступом пользователей к страницам приложения.

Серверная часть, реализованная на *Java* с использованием *Vaadin Backend*, предоставила стабильную и масштабируемую архитектуру для обработки пользовательских запросов, управления сессиями и взаимодействия с базой данных *PostgreSQL*. Вся бизнес-логика была распределена между слоями *entity*, *service* и *views*, что обеспечило логическую декомпозицию кода и упрощение его поддержки.

Для упрощения процесса развертывания и обеспечения изоляции компонентов была применена технология контейнеризации с использованием *Docker*. В рамках проекта использовались два контейнера: для серверной части и базы данных *PostgreSQL*, что позволило избежать конфликтов зависимостей и создать единую согласованную среду выполнения. Файл *docker-compose.yml* автоматизировал запуск компонентов и их взаимодействие, а использование томов для базы данных обеспечило сохранность данных при перезапуске контейнеров.

В целом, проект построен с учётом современных требований к безопасности, масштабируемости и простоте развёртывания. Организация клиентской и серверной частей в связке с контейнеризацией обеспечивает надёжную и стабильную работу приложения, упрощает внесение изменений и дальнейшее развитие функционала.

4 Тестирование приложения

4.1 Валидация форм ввода

В этом разделе подробно рассмотрим процесс тестирования завершенного приложения, сосредоточившись на клиентской части.

При регистрации пользователя в случае не ввода данных, будет выведено сообщение об ошибке. Вывод ошибки представлен на рисунке 4.1.

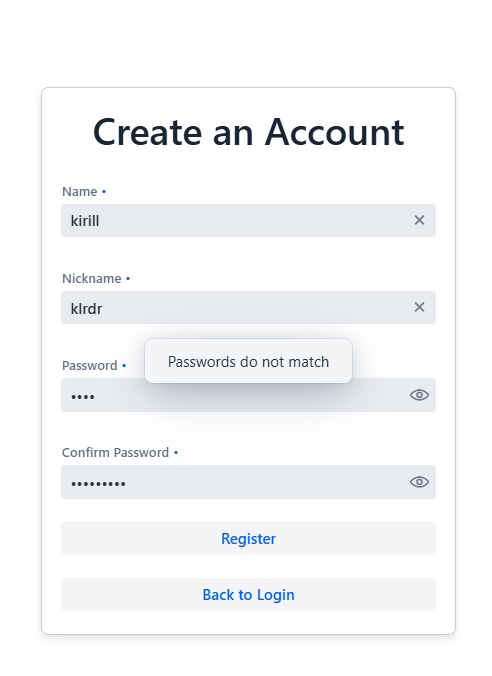


Рисунок 4.1 – Вывод ошибки некорректных данных при регистрации

При авторизации, если введённые данные не соответствуют зарегистрированным пользователям, авторизация не произойдёт. Поля отчистятся и подсветятся красным. Демонстрация на рисунке 4.2.

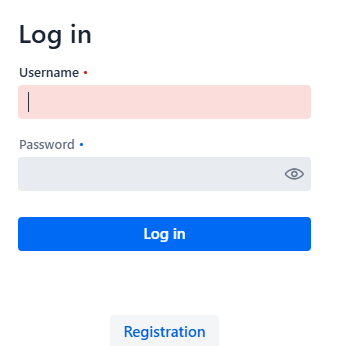


Рисунок 4.2 – Демонстрация неверности введения данных при авторизации

Как видно функционал построен так, чтобы не давать пользователям создавать незаполненные или неправильные объекты.

4.2 Проверка валидности ID комнаты видеоконференции

Для тестирования введём не валидный номер комнаты, пользователь получит сообщение о неверности введённого ID. Демонстрация рисунке 4.3.

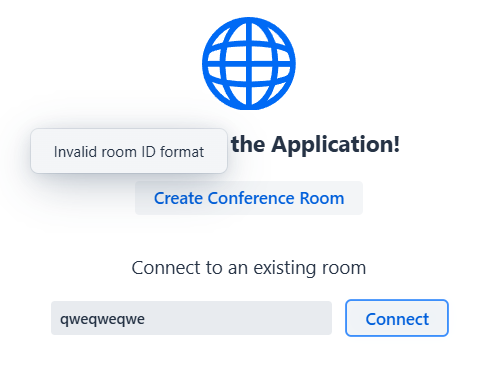


Рисунок 4.3 – Неверно введённый ID комнаты

Таким образом, система корректно обрабатывает ошибки ввода, информируя пользователя о некорректном идентификаторе комнаты и предотвращая доступ к видеоконференции.

4.3 Поиск несуществующих пользователей

При попытке поиска пользователя, которого нет в базе данных, система отображает соответствующее уведомление о том, что пользователь не найден. Это позволяет избежать путаницы и информирует пользователя о необходимости корректного ввода данных.

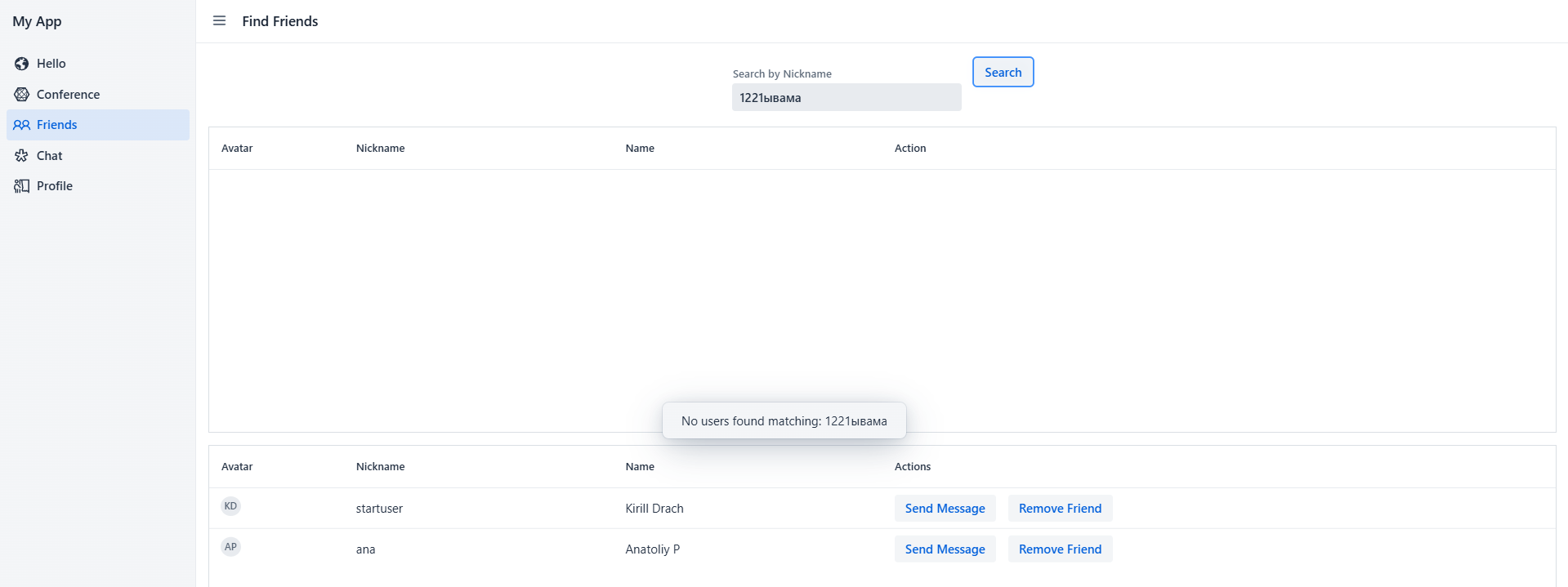


Рисунок 4.4 – Результат поиска пользователя

Таким образом, приложение обрабатывает ситуацию отсутствия пользователя в базе данных, поддерживая удобство и информативность интерфейса для пользователей.

4.4 Обработка сценария попытки создания комнаты гостем

Сценарий работы приложения предполагает создание комнат только зарегистрированным пользователем. В случае попытки создания комнаты гостем, получает сообщение о необходимости регистрации в сервисе. Продемонстрированно на рисунке 4.5.

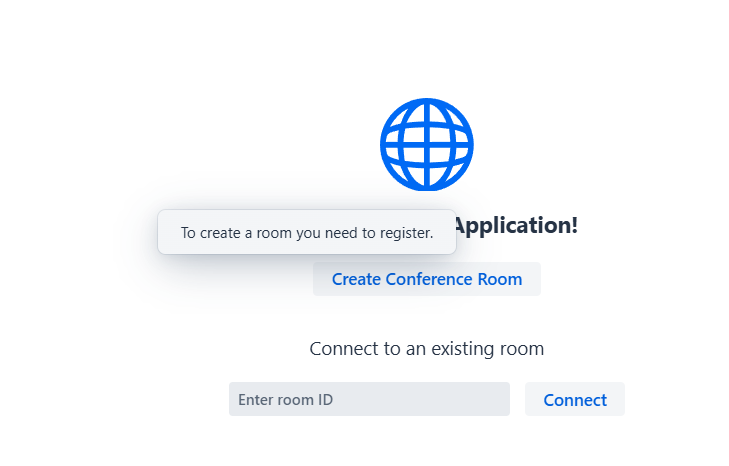


Рисунок 4.5 – Результат попытки создания комнаты гостем

После сообщения гостя перенаправит на страницу регистрации.

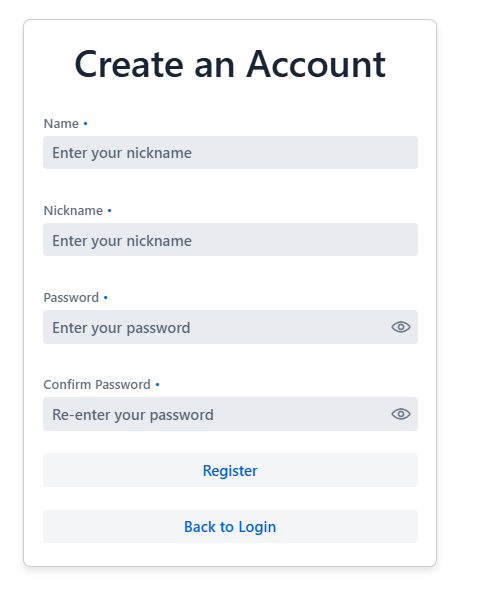


Рисунок 4.5 – Результат попытки создания комнаты гостем

Такой подход позволит увеличить количество зарегистрированных пользователей и повторное использование веб-приложения.

4.5 Вывод по разделу

В ходе тестирования клиентской части приложения было подтверждено её стабильное функционирование и соответствие заявленным требованиям. Все ключевые функции, включая регистрацию, авторизацию, создание и редактирование рабочих пространств, продемонстрировали корректную работу. Проверка

5 Руководство пользователя

5.1 Руководство пользователя

В данном разделе представлены инструкции по использованию приложения для бронирования рабочих мест и переговорных комнат для пользователя.

5.1.1 Регистрация пользователя

При регистрации пользователь должен ввести свой логин, почту, пароль и подтвердить его. Форма регистрации представлена на рисунке 5.1.

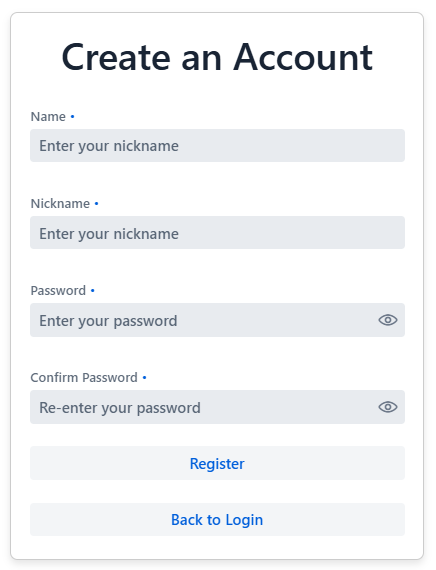


Рисунок 5.1 – Форма регистрации

После того, как пользователь нажмет на кнопку «Зарегистрироваться», будет отправлен запрос на регистрацию, на стороне сервера выполнится валидация всех полей. В случае ошибки, она будет выведена, а в случае успеха будет перенаправление на страницу авторизации. После чего менеджер должен будет активировать аккаунт.

5.1.2 Авторизация пользователя

На странице авторизации пользователь должен ввести уникальное свой логин и пароль, введённые при регистрации. Форма авторизации представлена на рисунке 5.2.

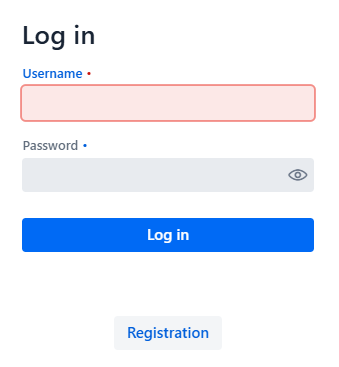


Рисунок 5.2 – Форма авторизации

В случае правильных учётных данных происходит перенаправление на главную страницу приложения.

5.1.3 Главная страница приложения

Попадая на главную страницу приложения, пользователь видит Приветствие. Главная страница приложения представлена на рисунке 5.3.

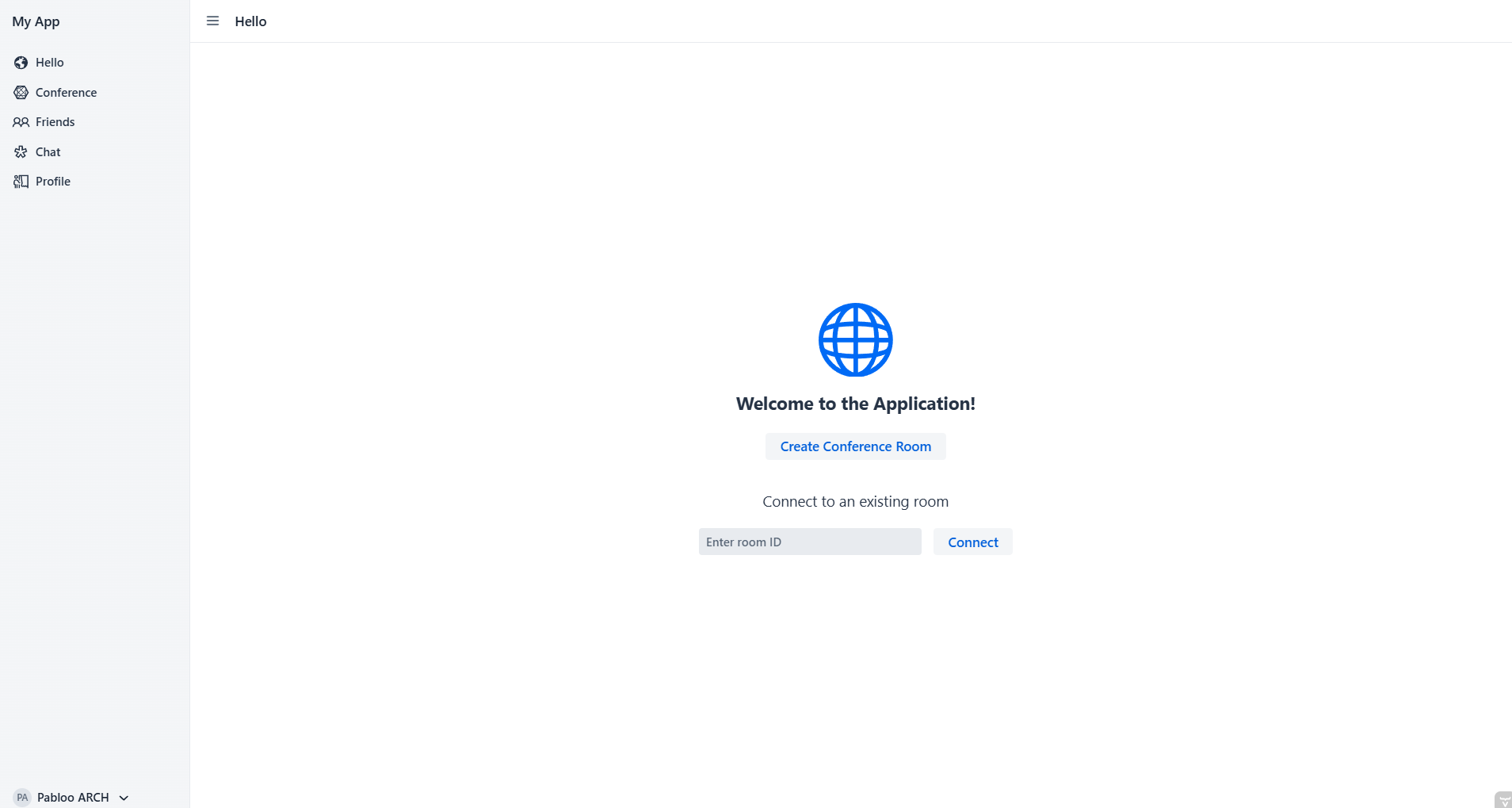


Рисунок 5.3 – Главная страница приложения

На главной странице пользователь может создать новую конференцию или подключиться к уже существующей, введя идентификатор комнаты. Удобная навигационная панель слева позволяет быстро перейти к ключевым разделам приложения, таким как профиль, список контактов, чат и конференции. Такой интерфейс обеспечивает интуитивное взаимодействие с основными функциями системы.

5.1.4 Подключение к видеоконференции

На главной странице пользователь может подключиться или создать комнату видеоконференции. Для создания можно нажать кнопку «Создать комнату», после чего пользователя перенаправит в комнату. Там он сможет скопировать пригласительную ссылку и поделится ею с другими пользователями.

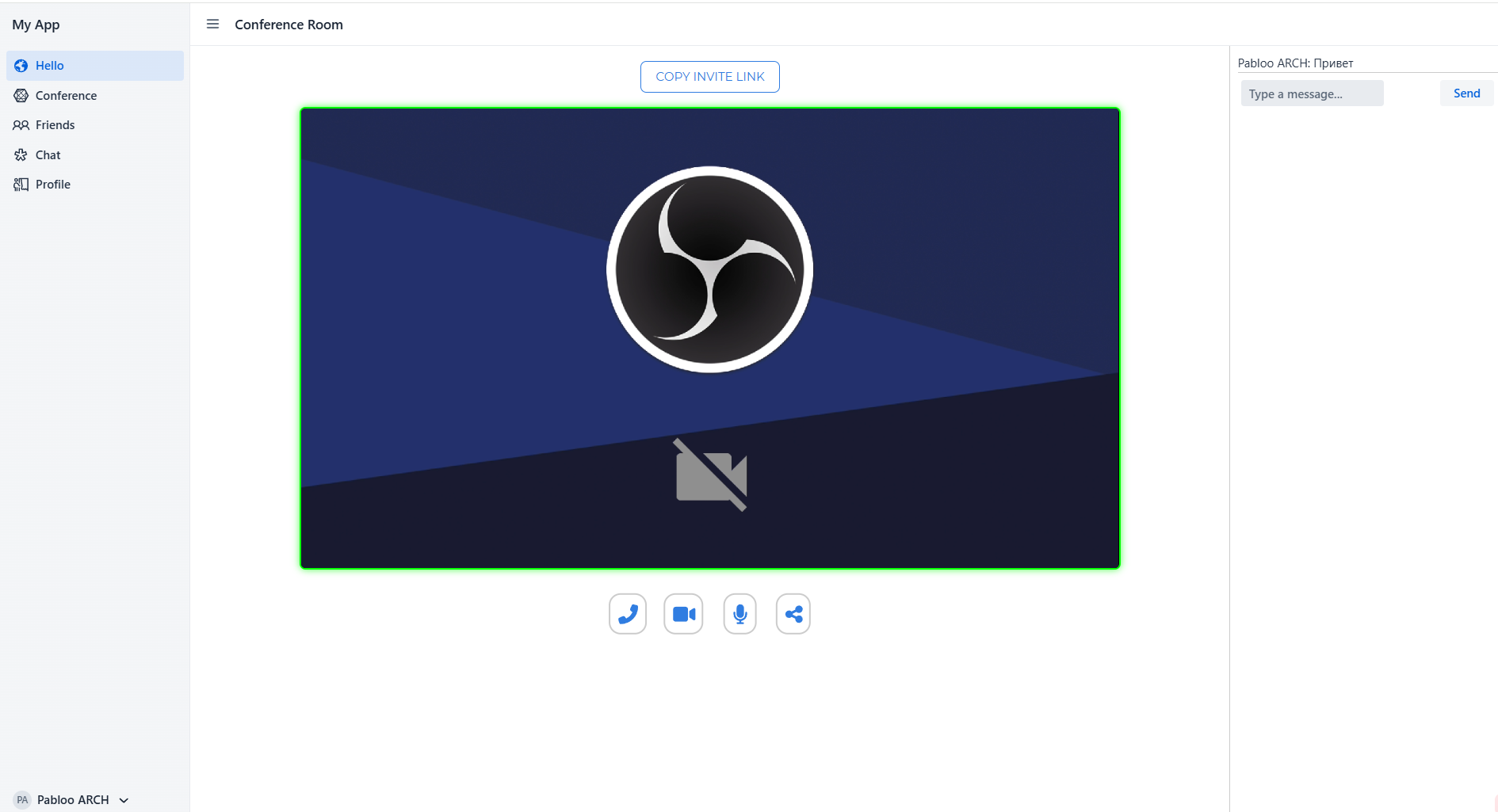


Рисунок 5.4 – Страница проведения видео конференции

Пользователь может отправлять сообщения в чат конференции и принимать их от других пользователей.

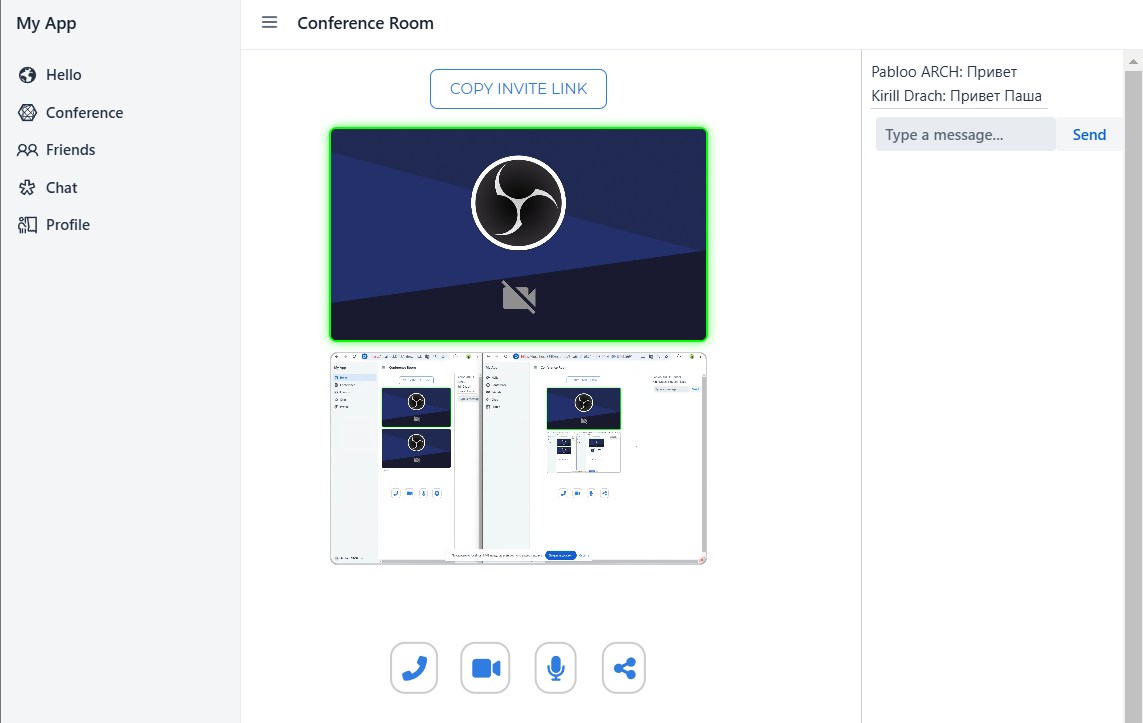


Рисунок 5.5 – Демонстрация использования чата конференции

Использование чата не является ограниченным.

5.1.5 Страница чата с пользователями

Если между пользователями установлен контакт, то они могут обмениваться сообщениями между друг другом. Демонстрация на рисунке 5.6.

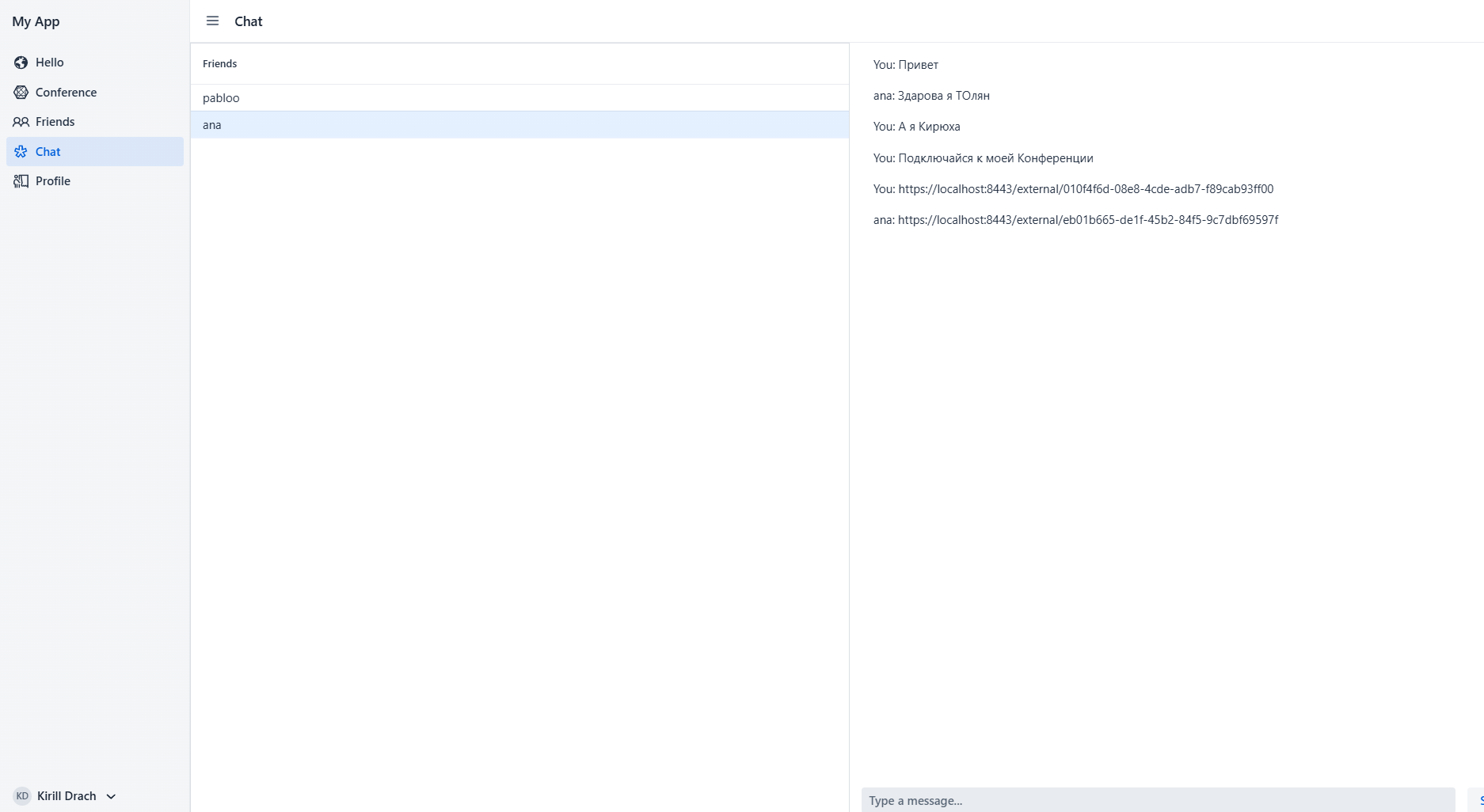


Рисунок 5.6 – Страница чата с пользователями

Если между пользователями установлен контакт, они получают возможность обмениваться сообщениями в режиме реального времени. Это обеспечивает удобное взаимодействие в рамках приложения и поддерживает эффективную коммуникацию.

5.1.6 Страница установления контактов

На странице установления контактов пользователи могут искать других участников по nickname с помощью строки поиска. Демонстрация на рисунке 5.7.

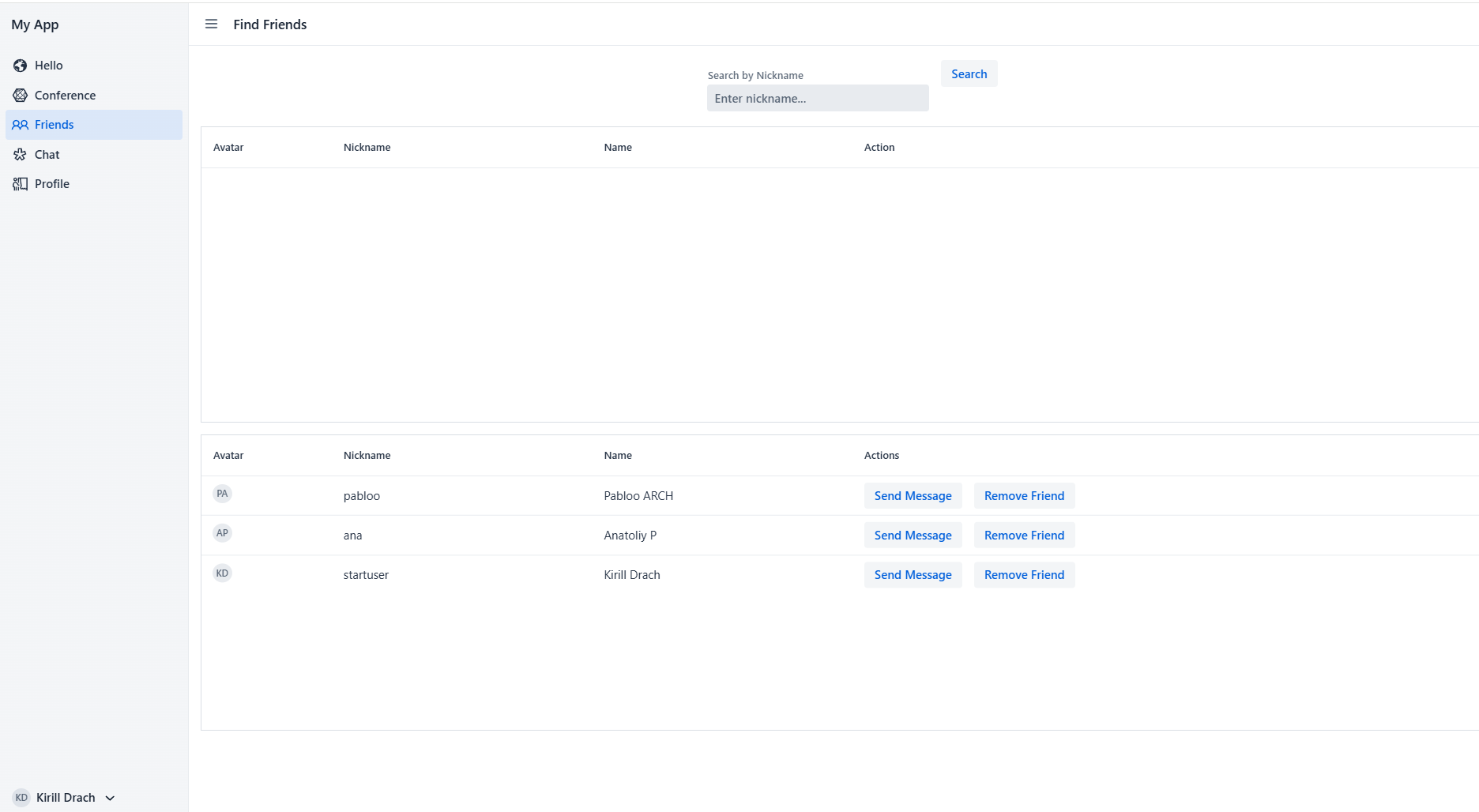


Рисунок 5.7 – Демонстрация страницы бронирования рабочих мест

При нахождении нужного пользователя в таблице отображается его nickname, имя и доступные действия — отправить сообщение (Send Message) или удалить из контактов (Remove Friend). Такая структура обеспечивает интуитивное управление контактами, позволяя пользователям быстро взаимодействовать друг с другом в приложении.

5.1.7 Комната видео конференции

На странице видеоконференции пользователи могут подключаться к виртуальной комнате для проведения встреч. Интерфейс предоставляет функции управления видеопотоком, включая включение и выключение камеры и микрофона, а также доступ к чату для обмена текстовыми сообщениями. Для подключения других участников можно использовать кнопку копирования пригласительной ссылки, что упрощает процесс добавления новых участников. Демонстрация на рисунке 5.8.

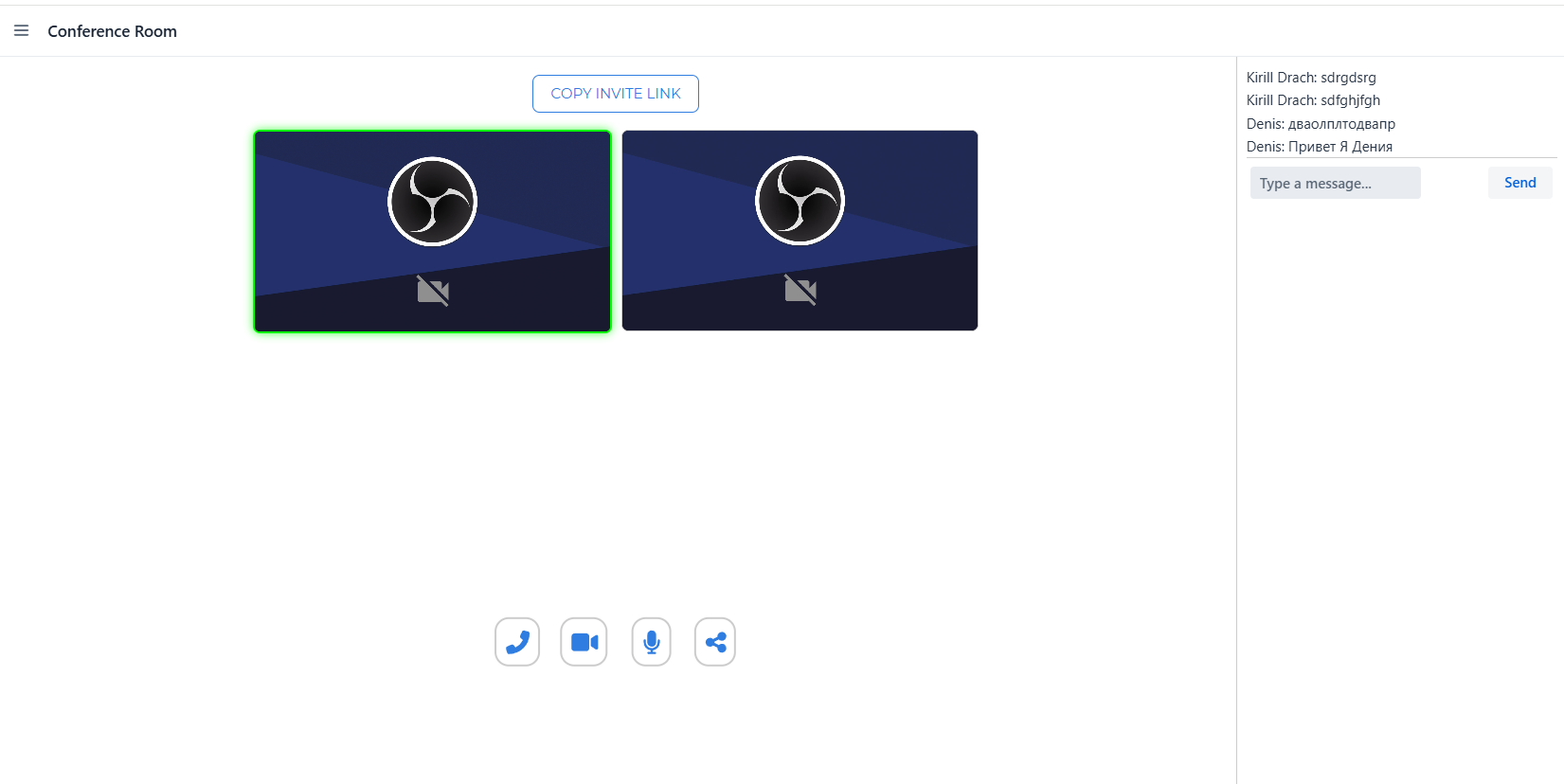


Рисунок 5.8 – Демонстрация комнаты видеоконференции

Как показано на рисунке, участники могут взаимодействовать друг с другом через видеопоток, а также использовать текстовый чат для передачи сообщений. Интерактивные элементы управления позволяют включать и отключать видео- и аудиопотоки, а также демонстрировать экран. Такой интерфейс делает процесс видеоконференции удобным и интуитивно понятным.

5.2 Руководство по развёртыванию приложения

В данном разделе представлена инструкция по развертыванию веб-приложения для видеоконференций с использованием Docker и файла конфигурации docker-compose.yml.

5.2.1 Подготовка к развёртыванию

Перед началом развёртывания необходимо выполнить следующие шаги:

* Установить *Docker* и *Docker Compose* на целевом устройстве;
* Убедиться, что файл *docker-compose.yml* и все необходимые директории проекта (включая сервер и базу данных) находятся в одном рабочем каталоге;
* Проверить наличие файла *.env* с необходимыми переменными окружения для работы приложения (например, данные для подключения к базе данных, порты и секреты).

5.2.2 Структура развёртывания

Конфигурация развертывания предусматривает использование двух контейнеров:

Контейнер базы данных запускается на основе образа postgres:15. Он настраивается через переменные окружения *POSTGRES\_DB*, *POSTGRES\_USER* и *POSTGRES\_PASSWORD*. Для обеспечения сохранности данных даже при перезапуске контейнера используется том *postgres\_data*. Контейнер открывает порт 5432, пробрасывая его на тот же порт хост-системы.

Приложение, объединяющее серверную и клиентскую части, создается на основе локального *Dockerfil*e. Перед запуском автоматически применяются миграции базы данных (*liquibase update*). Серверное приложение запускается на порту 8080, пробрасываемом на тот же порт хостовой системы. Внутри контейнера используется директория *./app* для хранения кода и данных.

Все контейнеры объединены в сеть *app-network*, что обеспечивает безопасное и стабильное взаимодействие между сервером и базой данных. Такой подход упрощает управление связями между компонентами системы, обеспечивая удобство масштабирования.

5.2.3 Запуск приложения

Для развертывания веб-приложения необходимо открыть терминал в корневой директории проекта и выполнить команду docker-compose up --build. Эта команда инициирует сборку образов и запуск контейнеров для всех компонентов системы. После успешного старта необходимо проверить доступность базы данных PostgreSQL на порту 5432, работу серверной части на порту 8080 и убедиться в корректном взаимодействии компонентов приложения.

5.3 Вывод по разделу

В данном разделе были подробно рассмотрены ключевые аспекты использования веб-приложения для организации видеоконференций, разработанного с использованием *Java, Vaadin, PostgreSQL* и *WebRTC*. Описан пользовательский функционал, включая подключение к конференции по *ID* комнаты, создание новых конференций, обмен сообщениями и управление контактами. Приложение предоставляет пользователям простой и интуитивно понятный интерфейс, где доступ к функциональным элементам реализован через основные вкладки: *Hello, Conference, Friends, Chat и Profile.*

Зарегистрированные пользователи могут выполнять такие действия, как отправка приглашений, создание новых конференций и управление контактами. Для гостей доступ ограничен подключением к конференции через уникальный ID комнаты или пригласительную ссылку, а для доступа к расширенным функциям, включая создание новых комнат и редактирование профиля, требуется регистрация. Использование протокола *WebRTC* обеспечивает передачу аудио и видео в реальном времени, что критически важно для видеосвязи высокого качества.

Техническая часть была организована с применением технологии контейнеризации с помощью *Docker*, что позволило изолировать серверное приложение и базу данных *PostgreSQL* в отдельных контейнерах. Это обеспечивает согласованность среды выполнения, упрощает развёртывание, обновление и масштабирование приложения. Подключение между контейнерами осуществляется через сеть, определённую в файле *docker-compose.yml*, что позволяет компонентам взаимодействовать через протокол *TCP*. Использование постоянного тома данных для *PostgreSQL* гарантирует сохранность информации о пользователях, конференциях и сообщениях даже при перезапуске контейнеров.

Таким образом, веб-приложение для организации видеоконференций демонстрирует надёжную архитектуру, высокую производительность и удобство использования как для гостей, так и для зарегистрированных пользователей. Технологии *Java* и *Vaadin* обеспечивают гибкость интерфейса, а *PostgreSQL* и *Docker* способствуют безопасности данных и лёгкости развертывания системы.

Заключение

В ходе выполнения курсового проекта было разработано веб-приложение для проведения видеоконференций, которое отвечает современным требованиям и предоставляет пользователям удобный и безопасный инструмент для онлайн-коммуникаций. Архитектура приложения, основанная на Java, Vaadin, PostgreSQL и WebRTC, была спроектирована с учетом принципов масштабируемости, надёжности и простоты использования, что делает систему удобной для дальнейшего расширения и поддержки.

В процессе разработки были успешно выполнены следующие задачи:

– Реализован безопасный механизм регистрации и авторизации, позволяющий разграничивать доступ для гостей и зарегистрированных пользователей.

– Внедрён функционал для проведения видеоконференций, включая подключение по ID комнаты и пригласительной ссылке, а также обмен текстовыми сообщениями во время конференции.

– Осуществлена поддержка системы контактов, позволяющая пользователям искать других участников, отправлять запросы на добавление в контакты и обмениваться личными сообщениями.

– Внедрена архитектура клиент-сервер с использованием Vaadin для построения интерфейса и серверной логики на Java, что обеспечило единую технологическую платформу.

– Выполнена интеграция базы данных PostgreSQL для надёжного хранения информации о пользователях, контактах, конференциях и сообщениях.

– Обеспечена контейнеризация с помощью Docker, что упрощает развертывание системы и её переносимость на разные платформы.

Технологический стек проекта, включающий Vaadin для интерфейса, Java для серверной части, PostgreSQL для базы данных и WebRTC для передачи аудио- и видеопотоков, позволил создать производительное, надёжное и интуитивно понятное приложение. Применение контейнеризации с Docker обеспечило изоляцию компонентов, стабильность окружения и удобство управления средой выполнения.

Итогом выполнения работы стало создание современного, безопасного и функционального веб-приложения для проведения видеоконференций, которое соответствует актуальным требованиям по производительности, удобству использования и надёжности. Приложение готово к дальнейшему расширению функционала и масштабированию, что делает его перспективным решением для использования в корпоративных и образовательных средах.

.

Список используемых источников

1. *React* документация. [Электронный ресурс]. Режим доступа: *https://reactjs.org/docs/getting-started.html*. Дата доступа: 15.09.2024.
2. *Django* документация. [Электронный ресурс]. Режим доступа: *https://docs.djangoproject.com/*. Дата доступа: 15.09.2024.
3. *Firebase* документация. [Электронный ресурс]. Режим доступа: *https://firebase.google.com/docs*. Дата доступа: 15.09.2024.
4. *Docker* документация. [Электронный ресурс]. Режим доступа: *https://docs.docker.com/*. Дата доступа: 16.09.2024.
5. *PostgreSQL* документация. [Электронный ресурс]. Режим доступа: *https://www.postgresql.org/docs/*. Дата доступа: 20.09.2024.
6. *Jitsi Meet* документация. [Электронный ресурс]. Режим доступа: *https://github.com/jitsi*. Дата доступа: 20.09.2024.
7. *Amazon Chime* документация. [Электронный ресурс]. Режим доступа: *https://aws.amazon.com/chime/*. Дата доступа: 20.09.2024.
8. *Zoho Meeting* документация. [Электронный ресурс]. Режим доступа: *https://www.zoho.com/meeting*/. Дата доступа: 20.09.2024.
9. *Amazon Chime* документация. [Электронный ресурс]. Режим доступа: *https://aws.amazon.com/chime/*. Дата доступа: 20.09.2024.

ПРИЛОЖЕНИЕ А

Листинг файла «AbstractEntity.java»

|  |
| --- |
| package com.example.application.data.entity;  import jakarta.persistence.GeneratedValue;  import jakarta.persistence.GenerationType;  import jakarta.persistence.Id;  import jakarta.persistence.MappedSuperclass;  import jakarta.persistence.SequenceGenerator;  import jakarta.persistence.Version;  @MappedSuperclass  public abstract class AbstractEntity {  @Id  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "idgenerator")  @SequenceGenerator(name = "idgenerator", initialValue = 1000)  private Long id;  @Version  private int version;  public Long getId() {  return id;  }  public void setId(Long id) {  this.id = id;  }  public int getVersion() {  return version;  }  @Override  public int hashCode() {  if (getId() != null) {  return getId().hashCode();  }  return super.hashCode();  }  @Override  public boolean equals(Object obj) {  if (!(obj instanceof AbstractEntity)) {  return false;  }  AbstractEntity other = (AbstractEntity) obj;  if (getId() != null) {  return getId().equals(other.getId());  }  return super.equals(other);  }  } |

ПРИЛОЖЕНИЕ Б

Листинг файла «MessageService.java»

|  |
| --- |
| package com.example.application.data.service;  import com.example.application.data.entity.Message;  import com.example.application.data.entity.User;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import java.time.LocalDateTime;  import java.util.List;  @Service  public class MessageService {  private final MessageRepository messageRepository;  public MessageService(MessageRepository messageRepository) {  this.messageRepository = messageRepository;  }  @Transactional  public Message sendMessage(User sender, User receiver, String content) {  Message message = new Message();  message.setSender(sender);  message.setReceiver(receiver);  message.setContent(content);  message.setSentAt(LocalDateTime.now());  return messageRepository.save(message);  }  public Message save(Message message) {  return messageRepository.save(message);  }  public List<Message> getMessagesBetweenUsers(User sender, User receiver) {  return messageRepository.findBySenderAndReceiverOrReceiverAndSenderOrderBySentAtAsc(sender, receiver, sender, receiver);  }  } |

ПРИЛОЖЕНИЕ В

Листинг файла «UserService.java»

|  |
| --- |
| package com.example.application.data.service;  import com.example.application.data.entity.User;  import org.springframework.data.domain.Page;  import org.springframework.data.domain.Pageable;  import org.springframework.data.jpa.domain.Specification;  import org.springframework.stereotype.Service;  import java.util.List;  import java.util.Optional;  @Service  public class UserService {  private final UserRepository repository;  private final FriendshipService friendshipService;  public UserService(UserRepository repository, FriendshipService friendshipService) {  this.repository = repository;  this.friendshipService = friendshipService;  }  public Optional<User> get(Long id) {  return repository.findById(id);  }  public User update(User entity) {  return repository.save(entity);  }  public void delete(Long id) {  repository.deleteById(id);  }  public Page<User> list(Pageable pageable) {  return repository.findAll(pageable);  }  public Page<User> list(Pageable pageable, Specification<User> filter) {  return repository.findAll(filter, pageable);  }  public int count() {  return (int) repository.count();  }  public Optional<User> getByUsername(String username) {  return Optional.ofNullable(repository.findByUsername(username));  }  public List<User> listAllByNicknameContains(String nickname) {  return repository.findByUsernameContainingIgnoreCase(nickname);  }  public List<User> getFriends(User user) {  return friendshipService.getFriends(user);  }  } |

ПРИЛОЖЕНИЕ Г

Листинг файла «MainLayout»

|  |
| --- |
| package com.example.application.views;  import com.example.application.components.appnav.AppNav;  import com.example.application.components.appnav.AppNavItem;  import com.example.application.data.entity.User;  import com.example.application.security.AuthenticatedUser;  import com.example.application.views.chat.ChatView;  import com.example.application.views.conference.ConferenceView;  import com.example.application.views.friends.FriendsView;  import com.example.application.views.hello.HelloView;  import com.example.application.views.user.ProfileView;  import com.vaadin.flow.component.applayout.AppLayout;  import com.vaadin.flow.component.applayout.DrawerToggle;  import com.vaadin.flow.component.avatar.Avatar;  import com.vaadin.flow.component.contextmenu.MenuItem;  import com.vaadin.flow.component.html.Anchor;  import com.vaadin.flow.component.html.Div;  import com.vaadin.flow.component.html.Footer;  import com.vaadin.flow.component.html.H1;  import com.vaadin.flow.component.html.H2;  import com.vaadin.flow.component.html.Header;  import com.vaadin.flow.component.icon.Icon;  import com.vaadin.flow.component.menubar.MenuBar;  import com.vaadin.flow.component.orderedlayout.Scroller;  import com.vaadin.flow.router.PageTitle;  import com.vaadin.flow.server.auth.AccessAnnotationChecker;  import com.vaadin.flow.theme.lumo.LumoUtility;  import java.util.Optional;  import org.vaadin.lineawesome.LineAwesomeIcon;  public class MainLayout extends AppLayout {  private H2 viewTitle;  private AuthenticatedUser authenticatedUser;  private AccessAnnotationChecker accessChecker;  public MainLayout(AuthenticatedUser authenticatedUser, AccessAnnotationChecker accessChecker) {  this.authenticatedUser = authenticatedUser;  this.accessChecker = accessChecker;  setPrimarySection(Section.DRAWER);  addDrawerContent();  addHeaderContent();  }  private void addHeaderContent() {  DrawerToggle toggle = new DrawerToggle();  toggle.getElement().setAttribute("aria-label", "Menu toggle");  viewTitle = new H2();  viewTitle.addClassNames(LumoUtility.FontSize.LARGE, LumoUtility.Margin.NONE);  addToNavbar(true, toggle, viewTitle);  }  private void addDrawerContent() {  H1 appName = new H1("My App");  appName.addClassNames(LumoUtility.FontSize.LARGE, LumoUtility.Margin.NONE);  Header header = new Header(appName);  Scroller scroller = new Scroller(createNavigation());  addToDrawer(header, scroller, createFooter());  }  private AppNav createNavigation() {  AppNav nav = new AppNav();  if (accessChecker.hasAccess(HelloView.class)) {  nav.addItem(new AppNavItem("Hello", HelloView.class, LineAwesomeIcon.GLOBE\_SOLID.create()));  }  if (accessChecker.hasAccess(ConferenceView.class)) {  nav.addItem(new AppNavItem("Conference ", ConferenceView.class, LineAwesomeIcon.CONNECTDEVELOP.create()));  }  if (accessChecker.hasAccess(FriendsView.class)) {  nav.addItem(new AppNavItem("Friends", FriendsView.class, LineAwesomeIcon.USER\_FRIENDS\_SOLID.create()));  }  if (accessChecker.hasAccess(ChatView.class)) {  nav.addItem(new AppNavItem("Chat", ChatView.class, LineAwesomeIcon.DIASPORA.create()));  }  if (accessChecker.hasAccess(ProfileView.class)) {  nav.addItem(new AppNavItem("Profile", ProfileView.class, LineAwesomeIcon.PERSON\_BOOTH\_SOLID.create()));  }  return nav;  }  private Footer createFooter() {  Footer layout = new Footer();  Optional<User> maybeUser = authenticatedUser.get();  if (maybeUser.isPresent()) {  User user = maybeUser.get();  Avatar avatar = new Avatar(user.getName());  avatar.setThemeName("xsmall");  avatar.getElement().setAttribute("tabindex", "-1");  MenuBar userMenu = new MenuBar();  userMenu.setThemeName("tertiary-inline contrast");  MenuItem userName = userMenu.addItem("");  Div div = new Div();  div.add(avatar);  div.add(user.getName());  div.add(new Icon("lumo", "dropdown"));  div.getElement().getStyle().set("display", "flex");  div.getElement().getStyle().set("align-items", "center");  div.getElement().getStyle().set("gap", "var(--lumo-space-s)");  userName.add(div);  userName.getSubMenu().addItem("Sign out", e -> {  authenticatedUser.logout();  });  layout.add(userMenu);  } else {  Anchor loginLink = new Anchor("login", "Sign in");  layout.add(loginLink);  }  return layout;  }  @Override  protected void afterNavigation() {  super.afterNavigation();  viewTitle.setText(getCurrentPageTitle());  }  private String getCurrentPageTitle() {  PageTitle title = getContent().getClass().getAnnotation(PageTitle.class);  return title == null ? "" : title.value();  }  } |

ПРИЛОЖЕНИЕ Д

Листинг файла «FriendsView.java»

|  |
| --- |
| package com.example.application.views.friends;  import com.example.application.data.entity.Friendship;  import com.example.application.data.entity.FriendshipStatus;  import com.example.application.data.entity.User;  import com.example.application.data.service.FriendshipService;  import com.example.application.data.service.UserService;  import com.example.application.utils.ChatStateService;  import com.example.application.views.MainLayout;  import com.example.application.data.service.CurrentUser;  import com.example.application.views.chat.ChatView;  import com.vaadin.flow.component.avatar.Avatar;  import com.vaadin.flow.component.button.Button;  import com.vaadin.flow.component.grid.Grid;  import com.vaadin.flow.component.notification.Notification;  import com.vaadin.flow.component.orderedlayout.HorizontalLayout;  import com.vaadin.flow.component.orderedlayout.VerticalLayout;  import com.vaadin.flow.component.textfield.TextField;  import com.vaadin.flow.router.PageTitle;  import com.vaadin.flow.router.Route;  import com.vaadin.flow.server.StreamResource;  import jakarta.annotation.security.PermitAll;  import org.springframework.beans.factory.annotation.Autowired;  import java.io.ByteArrayInputStream;  import java.util.List;  @PageTitle("Find Friends")  @Route(value = "friends", layout = MainLayout.class)  @PermitAll  public class FriendsView extends VerticalLayout {  private final UserService userService;  private final FriendshipService friendshipService;  private final CurrentUser currentUser;  private final Grid<User> userGrid = new Grid<>(User.class, false);  private final Grid<User> friendsGrid = new Grid<>(User.class, false);  private final TextField searchField = new TextField("Search by Nickname");  @Autowired  public FriendsView(UserService userService, FriendshipService friendshipService, CurrentUser currentUser) {  this.userService = userService;  this.friendshipService = friendshipService;  this.currentUser = currentUser;  setAlignItems(Alignment.CENTER);  setJustifyContentMode(JustifyContentMode.CENTER);  setSpacing(true);  searchField.setPlaceholder("Enter nickname...");  searchField.setWidth("300px");  Button searchButton = new Button("Search", event -> searchUsers());  HorizontalLayout searchLayout = new HorizontalLayout(searchField, searchButton);  configureUserGrid();  configureFriendsGrid();  add(searchLayout, userGrid, friendsGrid);  loadFriends();  }  private void configureUserGrid() {  userGrid.addComponentColumn(user -> {  Avatar avatar = new Avatar(user.getName());  if (user.getProfilePicture() != null) {  StreamResource resource = new StreamResource("profile-pic",  () -> new ByteArrayInputStream(user.getProfilePicture()));  avatar.setImageResource(resource);  }  avatar.setThemeName("xsmall");  return avatar;  }).setHeader("Avatar").setWidth("10%").setFlexGrow(0);  userGrid.addColumn(User::getUsername)  .setHeader("Nickname")  .setWidth("20%").setFlexGrow(0);  userGrid.addColumn(User::getName)  .setHeader("Name")  .setWidth("20%").setFlexGrow(0);  userGrid.addComponentColumn(user -> {  Button addFriendButton = new Button("Add Friend", event -> sendFriendRequest(user));  addFriendButton.setEnabled(!isAlreadyFriend(user));  return addFriendButton;  }).setHeader("Action").setWidth("50%").setFlexGrow(0);  }  private void configureFriendsGrid() {  friendsGrid.addComponentColumn(user -> {  Avatar avatar = new Avatar(user.getName());  if (user.getProfilePicture() != null) {  StreamResource resource = new StreamResource("profile-pic",  () -> new ByteArrayInputStream(user.getProfilePicture()));  avatar.setImageResource(resource);  }  avatar.setThemeName("xsmall");  return avatar;  }).setHeader("Avatar").setWidth("10%").setFlexGrow(0);  friendsGrid.addColumn(User::getUsername)  .setHeader("Nickname")  .setWidth("20%").setFlexGrow(0);  friendsGrid.addColumn(User::getName)  .setHeader("Name")  .setWidth("20%").setFlexGrow(0);  friendsGrid.addComponentColumn(user -> {  HorizontalLayout actionButtons = new HorizontalLayout();  if (isPendingRequest(user)) {  Button acceptButton = new Button("Add Friend", event -> acceptFriendRequest(user));  Button rejectButton = new Button("Reject", event -> rejectFriendRequest(user));  actionButtons.add(acceptButton, rejectButton);  } else if (isFriend(user)) {  Button sendMessageButton = new Button("Send Message", event -> sendMessage(user));  Button removeFriendButton = new Button("Remove Friend", event -> removeFriend(user));  actionButtons.add(sendMessageButton, removeFriendButton);  }  return actionButtons;  }).setHeader("Actions").setWidth("50%").setFlexGrow(0);  }  private boolean isPendingRequest(User user) {  return currentUser.get()  .map(current -> friendshipService.isFriendRequestPending(user, current))  .orElse(false);  }  private boolean isFriend(User user) {  return currentUser.get()  .map(current -> friendshipService.getFriends(current).contains(user))  .orElse(false);  }  private void acceptFriendRequest(User user) {  currentUser.get().ifPresent(receiver -> {  friendshipService.updateFriendshipStatus(user, receiver, FriendshipStatus.ACCEPTED);  Notification.show("Friend request accepted from " + user.getUsername(), 2000, Notification.Position.MIDDLE);  loadFriends();  });  }  private void rejectFriendRequest(User user) {  currentUser.get().ifPresent(receiver -> {  friendshipService.rejectFriendRequest(user, receiver);  Notification.show("Friend request rejected from " + user.getUsername(), 2000, Notification.Position.MIDDLE);  loadFriends();  });  }  private void removeFriend(User user) {  currentUser.get().ifPresent(current -> {  friendshipService.removeFriendship(current, user);  Notification.show("Removed friend: " + user.getUsername(), 2000, Notification.Position.MIDDLE);  loadFriends();  });  }  @Autowired  private ChatStateService chatStateService;  private void sendMessage(User user) {  chatStateService.setSelectedFriend(user);  getUI().ifPresent(ui -> ui.navigate(ChatView.class));  }  private void searchUsers() {  String searchTerm = searchField.getValue().trim();  if (searchTerm.isEmpty()) {  Notification.show("Please enter a nickname to search.", 2000, Notification.Position.MIDDLE);  return;  }  List<User> results = userService.listAllByNicknameContains(searchTerm);  userGrid.setItems(results);  if (results.isEmpty()) {  Notification.show("No users found matching: " + searchTerm, 2000, Notification.Position.MIDDLE);  }  }  private void sendFriendRequest(User user) {  currentUser.get().ifPresent(requester -> {  if (!friendshipService.isFriendRequestPending(requester, user)) {  friendshipService.sendFriendRequest(requester, user);  Notification.show("Friend request sent to " + user.getUsername(), 2000, Notification.Position.MIDDLE);  searchUsers();  } else {  Notification.show("Friend request already sent to " + user.getUsername(), 2000, Notification.Position.MIDDLE);  }  });  }  private void loadFriends() {  currentUser.get().ifPresent(user -> {  List<User> friendsAndRequests = friendshipService.getFriendsAndPendingRequests(user);  friendsGrid.setItems(friendsAndRequests);  });  }  private boolean isAlreadyFriend(User user) {  return currentUser.get().map(current -> {  boolean isFriend = friendshipService.getFriends(current).contains(user);  boolean requestAlreadySent = friendshipService.isFriendRequestPending(current, user);  return isFriend || requestAlreadySent;  }).orElse(false);  }  } |

ПРИЛОЖЕНИЕ Е

Листинг файла «docker-compose.yml»

|  |
| --- |
| version: '3.8'  services:  db:  image: postgres:15  environment:  POSTGRES\_DB: postgres  POSTGRES\_USER: postgres  POSTGRES\_PASSWORD: PB1234PB  ports:  - "5432:5432"  volumes:  - postgres\_data:/var/lib/postgresql/data  networks:  - app-network  vaadin-backend:  build:  context: ./backend  dockerfile: Dockerfile  environment:  DB\_HOST: db  DB\_PORT: 5432  DB\_NAME: postgres  DB\_USER: postgres  DB\_PASSWORD: PB1234PB  ports:  - "8080:8080"  volumes:  - ./backend:/app  working\_dir: /app  depends\_on:  - db  networks:  - app-network  webrtc-signaling:  build:  context: ./signaling  dockerfile: Dockerfile  ports:  - "8081:8081"  environment:  DB\_HOST: db  DB\_PORT: 5432  DB\_NAME: postgres  DB\_USER: postgres  DB\_PASSWORD: PB1234PB  networks:  - app-network  depends\_on:  - db  networks:  app-network:  driver: bridge  volumes:  postgres\_data:  external: false |