

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**«Реализация Web-платформы для изучения
иностраннных языков»**

Студент

(подпись, дата)

Тимофеев Егор Кириллович
(ФИО полностью)

Руководитель

(подпись, дата)

Кольчугина Е.А.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов «25»
декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Тимофеев Егор Кириллович гр. 20ВИ1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы «Реализация Web-платформы для изучения иностранных языков»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы Кольчугина Е.А.

3. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение: разработка и внедрение веб-приложения, предназначенного для
организации взаимодействия пользователей посредством обмена сообщениями в режиме
реального времени.

Основные функции системы:

1. Управление учётными данными пользователей;
2. Обеспечение возможность обмена текстовыми сообщениями;
3. Обеспечение возможность общения по видеосвязи;
4. Настройка фильтров по изучаемым языкам и рейтингу пользователей.

Технология разработки: объектно-ориентированное программирование (ООП),
архитектурный стиль клиент-сервер.

Средства разработки: Javascript, React, Node.js.

Среда разработки: WebStorm.

4. Перечень подлежащих разработке вопросов _____

1. Анализ предметной области и требований к системе. _____

2. Проектирование программных средств. _____

3. Реализация программных средств. _____

4. Тестирование программных средств. _____

5. Оформление пояснительной записки. _____

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 89 листов, 26 рисунков, 12 таблиц, 8 использованных источников и 2 приложения.

WEB-ПРИЛОЖЕНИЕ, ИЗУЧЕНИЕ ИНОСТРАННЫХ ЯЗЫКОВ, UML, REACT, NODE.JS, EXPRESS, IDEF1X, MS SQL, PLAYWRIGHT

Предмет работы: разработка Web-платформы для изучения иностранных языков.

Цель данной работы: разработка Web-платформы, позволяющей пользователям взаимодействовать друг с другом для изучения иностранных языков.

Средства разработки: React, Node.js, MS SQL.

Результаты работы: спроектировано и разработано Web-платформа для изучения иностранных языков.

Область применения: изучение иностранных языков посредством текстового, а также аудио-визуального общения.

					ПГУ 09.03.02 – 8ВР201.14 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Web-платформа для изучения иностранных языков» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Тимофеев Е.К.						
Провер.		Кольчугина Е.А.					4	89
Н. Контр.		Такташкин Д.В.				группа 20ВИ1		
Утверд.								

Содержание

Введение.....	6
1. Постановка задачи и анализ аналогов разрабатываемой системы.....	7
1.1 Постановка задачи.....	7
1.2 Анализ информационных систем аналогичного назначения	8
2. Выбор технологии проектирования и инструментальных средств	14
3. Проектирование структуры данных.....	16
3.1 Диаграмма классов.....	16
3.2 Диаграмма IDEF1x	21
4. Проектирование Web-приложения.....	23
4.1. Диаграмма вариантов использования	24
4.2 Диаграмма деятельности	25
4.3 Диаграмма состояний	27
5. Проектирование пользовательского интерфейса.....	31
6. Проведение функционально-стоимостного анализа проекта.....	37
7. Программная реализация.....	40
7.1 Реализация архитектуры	40
7.2 Реализация кода Web-приложения.....	47
8. Тестирование Web-приложения	51
8.1 Выбор и реализация метода тестирования	51
8.2 Функциональное тестирование.....	52
8.3 Тестирование совместимости	55
8.4 Тестирование пользовательского интерфейса	56
8.5 Расчет метрик кода.....	57
Заключение	61
Список используемых источников.....	62
Приложение А_Листинг программы	64
Приложение Б Результаты тестирования	84

Введение

Современные технологии революционизировали подход к изучению иностранных языков. Ранее основными методами обучения были традиционные очные занятия и учебники, но в наши дни все большую популярность приобретают онлайн-курсы и мобильные приложения. Web-приложения для изучения иностранных языков предоставляют уникальные возможности для обучения: доступ к разнообразным учебным материалам, интерактивные упражнения и автоматизированные системы проверки знаний.

Целью данной дипломной работы является разработка web-платформы для изучения иностранных языков, которая будет удовлетворять потребности пользователей разного уровня подготовки и возраста. Приложение должно быть интуитивно понятным, функциональным и эффективным для повышения уровня владения языком. Введение таких приложений способствует не только ускорению и упрощению процесса обучения, но и делает его более интересным и доступным для широкой аудитории.

В данной работе рассматриваются основные этапы разработки web-приложения, включая анализ предметной области, постановку задачи, выбор технологий, реализацию, тестирование и оценку эффективности. Особое внимание уделяется интерактивности приложения и адаптивности под нужды пользователей.

Предмет работы: Разработка Веб-платформы для изучения иностранных языков.

1. Постановка задачи и анализ аналогов разрабатываемой системы

1.1 Постановка задачи

Изучение иностранных языков с использованием web-приложений имеет множество преимуществ. Такой метод изучения обеспечивает гибкость в выборе времени и места обучения, что особенно важно для занятых людей, а интерактивные методы обучения, такие как игры, викторины, видео- и аудиоуроки, способствуют более глубокому усвоению материала, возможность автоматической проверки знаний позволяет оперативно выявлять и исправлять ошибки, что повышает эффективность обучения.

На рынке существует множество приложений для изучения языков, предлагающих различные функции: от базового обучения лексике и грамматике до сложных систем с элементами искусственного интеллекта для персонализации обучения. Однако многие из них либо слишком сложны в использовании, либо предлагают недостаточный функционал для полноценного изучения языка.

Веб-приложение должно обеспечивать выполнение следующих основных функций:

- **Управление** учётными данными пользователей;
- Обеспечить возможность обмена текстовыми сообщениями;
- Обеспечить возможность общения по видеосвязи;
- Настройка фильтров по изучаемым языкам и рейтингу пользователей.

Пользователь вводит свои учетные данные для входа в систему. Приложение проверяет наличие учетной записи и правильность введенных данных. В случае успешной проверки пользователь получает доступ к своему личному кабинету, где может управлять своими данными и настройками, а также видит свой рейтинг.

После входа в систему пользователи могут обмениваться текстовыми сообщениями. Для этого приложение создает соединение с сервером, используя

протоколы транспортного уровня TCP. Если соединение не установлено, на панели управления отображается сообщение об ошибке. Если соединение успешно установлено, создаются TCP сокеты для передачи и приема сообщений. Пользователь может отправлять и получать сообщения в режиме реального времени.

Для обеспечения видеосвязи приложение использует протокол RTP. Пользователь выбирает контакт для видеозвонка, и приложение проверяет наличие подключения к клиенту. В случае успешного подключения создаются RTP сокеты для передачи аудио- и видеопотоков. Пользователь нажимает кнопку для начала видеозвонка, после чего начинается потоковая передача данных.

В приложении предусмотрены настройки фильтров, которые позволяют пользователям выбирать интересующие их языки и фильтровать других пользователей по рейтингу. Это позволяет находить подходящих партнеров для общения и совместного изучения языков. Пользователь выбирает параметры фильтрации и применяет их, после чего обновляется список доступных пользователей.

В результате реализации данных функций, веб-приложение обеспечивает эффективное и удобное взаимодействие пользователей для изучения иностранных языков, предлагая различные инструменты для общения и обучения.

1.2 Анализ информационных систем аналогичного назначения

Перед началом разработки web-приложения для изучения иностранных языков необходимо провести анализ существующих систем, чтобы учесть их преимущества и недостатки. На рынке существует множество приложений для изучения языков, отличающихся по функциональности, технической реализации, стоимости и другим параметрам. Как показал анализ, для разработки web-приложения необходимо учитывать следующие аспекты:

- архитектура создаваемого приложения;
- принципы его работы;

- информация о возможностях приложения и услугах;
- способы отображения и навигации по контенту;
- ссылки на контактную информацию и сведения о разработчиках;
- методы взаимодействия пользователей.

Рассмотрим несколько существующих аналогов разрабатываемого web-приложения: Tandem, HelloTalk и Chatroulette.

Tandem – это приложение для языкового обмена, которое связывает пользователей, желающих изучать языки, с носителями этих языков. Платформа предоставляет возможности для общения через текст, голосовые и видеосообщения.

Приложение имеет следующие особенности:

- коррекция **текста: Функция** для исправления текстов, написанных вашим партнером, что помогает улучшить навыки письма;
- темы разговора: Приложение предлагает различные темы для разговора, чтобы сделать общение более интересным и структурированным;
- профессиональные уроки: Возможность брать уроки у профессиональных преподавателей.

Интерфейс приложения представлен на рисунке 1.

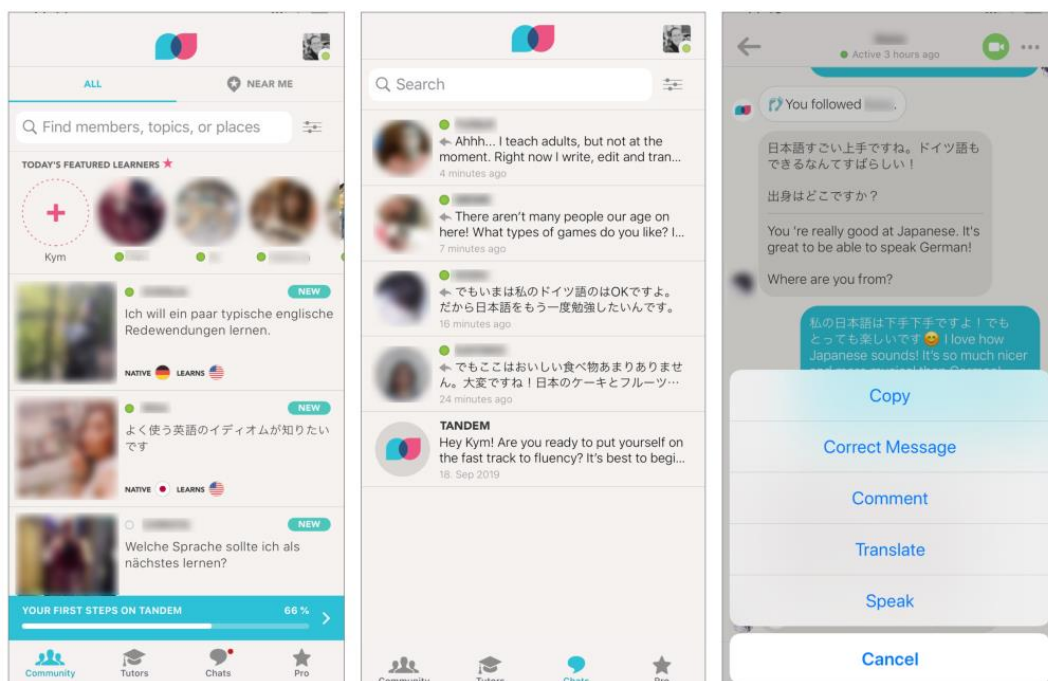


Рисунок 1 – Интерфейс приложения Tandem

HelloTalk – ещё одна популярная социальная платформа для языкового обмена, которая предлагает множество инструментов для практики языка, изучения языка. Также приложение предоставляет возможность заниматься с преподавателями языка и участвовать в различных сообществах внутри данной социальной платформы.

Лозунг приложения: «Лучший способ для изучения языка – это разговаривать на нем!».

Особенности приложения включают:

- перевод и исправление: Интегрированные функции перевода и исправления текста;
- инструменты для обучения: Функции для записи и прослушивания произношения, возможность добавлять исправления и комментарии к тексту;
- сообщество: Групповые чаты и обсуждения для практики языка в коллективе;
- профессиональные преподаватели: Доступ к профессиональным урокам с преподавателями;
- социальная платформа: Возможность публиковать «моменты» (подобно статусам или постам в социальных сетях), делиться фотографиями и видео;
- модерация: при создании аккаунта происходит модерация аккаунтов для предотвращения нарушений правил платформы;
- безопасность: проверка профилей для обеспечения безопасности.

Интерфейс мобильного приложения напоминает мессенджер с множеством активных вкладок. Это говорит о многофункциональности платформы, но может быть непонятно для новых пользователей.

Интерфейс платформы представлен на рисунке 2.

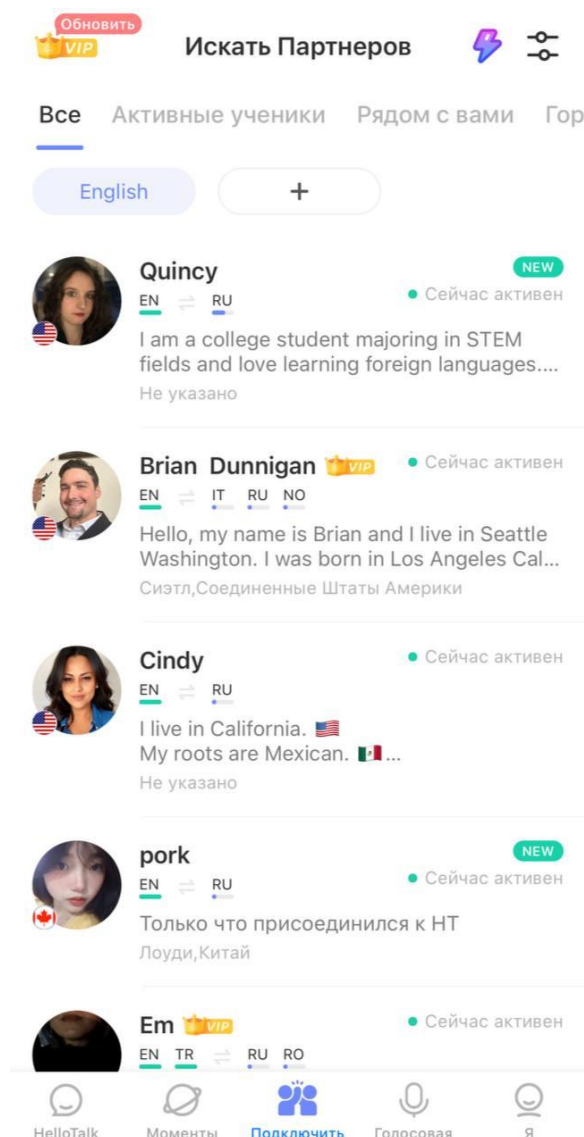


Рисунок 2 – Интерфейс приложения HelloTalk

Chatroulette – это платформа для случайных видеочатов, которая связывает пользователей со случайными собеседниками по всему миру. Основной фокус – на спонтанное и анонимное общение. Хотя Chatroulette может использоваться для практики языков, его основная цель – случайные видеочаты. Платформа не предоставляет специальных образовательных инструментов и функций, и взаимодействие на ней менее структурировано по сравнению с Tandem и HelloTalk. Интерфейс приложения представлен на рисунке 3.

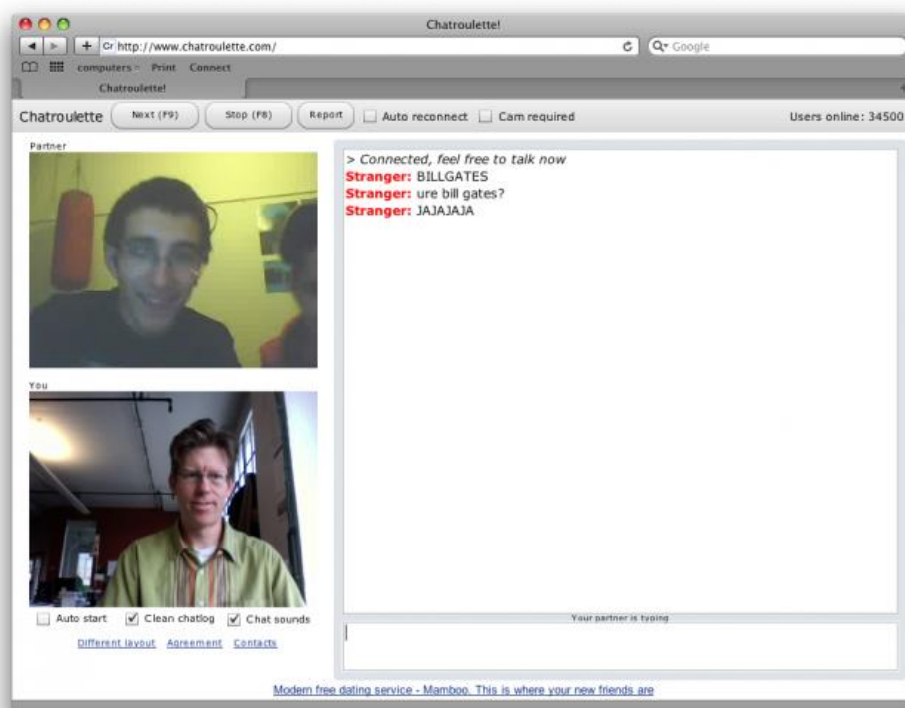


Рисунок 3 – Интерфейс приложения Chatroulette

Рассмотрев функциональные характеристики каждого аналога, был приведен сравнительный анализ, результаты которого приведены в таблице 1.

Таблица 1 – Сравнительный анализ систем-аналогов

Функциональные возможности	Tandem	HelloTalk	Chatroulette	Разрабатываемая система
Видеосвязь	Да	Да	Да	Да
Голосовые сообщения	Да	Да	Нет	Нет
Текстовые сообщения	Да	Да	Да	Да
Личный кабинет пользователя	Да	Да	Нет	Да

Продолжение таблицы 1

Платформы	Мобильное приложение	Мобильное приложение	Веб-приложение	Веб-приложение
Платные возможности	Да	Да	Да	Нет
Публикации	Нет	Да	Нет	Нет
Анонимное использование	Нет	Нет	Да	Нет
Добавление темы для разговора	Да	Нет	Нет	Нет
Фильтр собеседника по языку	Да	Да	Нет	Да
Отображаемый рейтинг пользователей	Нет	Нет	Нет	Да

В ходе анализа было принято решение разработать веб-приложение с поддержкой пользовательского интерфейса, чтобы у пользователя был доступ в систему компьютера, имеющий доступ в интернет.

Во всех системах аналогичного назначения часть функционала ограничена при бесплатном использовании, поэтому было принято решение не создавать подобного ограничения, чтобы разрабатываемая платформа выгодно отличалась на фоне конкурентов.

Также было принято решение сделать обязательную регистрацию для пользователей, чтобы обеспечивать безопасность и возможность комфортного общения всех пользователей. Помимо этого, было решено добавить функционал выбора тем для разговора, так как этот функционал поможет новым пользователям, но реализован только в одной из аналогичных систем.

2. Выбор технологии проектирования и инструментальных средств

Для проектирования веб-приложения для изучения иностранных языков магазина были выбраны несколько инструментов, которые вместе обеспечат полный цикл создания внедрения системы.

Draw.io был выбран для построения UML-диаграмм и других визуальных средств представления данных, благодаря своему широкому функционалу, а также возможности использовать этот инструмент бесплатно внутри любого браузера. Erwin Data Modeler, в свою очередь, предоставляет возможности для моделирования данных, необходимых для эффективной работы в области управления информацией.

Erwin Data Modeler – программа для моделирования данных, которая предоставляет возможность проектирования БД, включая логические, физические, архитектурные и сложные сценарии интеграции данных, необходимые для эффективной работы в области управления данными.

При выборе инструментов учитывались такие факторы, как наличие необходимого функционала, удобство использования и пользовательский интерфейс. Эти факторы имеют важное значение для успешной разработки информационной системы.

Для разработки приложения были выбраны следующие технологии: язык программирования JavaScript, библиотека React, интегрированная среда разработки WebStorm и модуль npm для Node.js.

JavaScript – удивительный multifunctional язык, широко используемый, кроме всего прочего, веб-разработке. Любое действие на веб-странице – это работа JavaScript. Фактически все современные браузеры могут взаимодействовать с JavaScript. Он определяет, что может делать страница и как она будет взаимодействовать с пользователем или серверной частью.

Веб-страница – это не только JavaScript. Она создается с помощью трех инструментов: HTML, CSS и JavaScript. HTML определяет то, что отображается

на странице: ее содержимое хранится в нем, состоит из элементов, которые называются тегами. Они описывают конкретный компонент страницы.

CSS-это шаблон веб-страницы. Размер шрифта, семейство шрифтов и расположение текста и объектов на странице задаются с его помощью.

React – это JavaScript библиотека, которая используется для создания пользовательских интерфейсов веб-приложений. React работает на основе компонентного подхода, что делает разработку более модульной и удобной. Она основывается на принципах обновления данных в режиме реального времени и взаимодействия с пользователем через события и обратные вызовы. React также помогает ускорить время разработки и улучшить производительность приложений, так как использует DOM (Document Object Model), которая позволяет обновить только те страницы, которые действительно изменились. Это увеличивает производительность приложения и уменьшает количество запросов к серверу.

В качестве инструмента для установки и настройки приложения на сервере принято решение использовать Node.js. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода, подключать разные пакеты и внешние библиотеки, обеспечивая вызовы к ним из JavaScript-кода.

Выбранные технологии разработки и проектирования информационных систем позволяет с успехом выполнить необходимую работу, одновременно являясь при этом удобными и эффективными в использовании.

3. Проектирование структуры данных

Проектирование структуры данных было выполнено с использованием Case-средства ERwin Data Modeler. ERwin представляет собой одно из CASE-средств, которое позволяет моделировать бизнес-процессы. Модели данных, создаваемые с помощью ERwin Data Modeler, подразделяются на две категории:

- логические модели, предназначенные для представления элементов в терминах бизнес-процессов (для моделирования используются диаграммы «сущность-связь»);
- физические модели, предназначенные для создания баз данных информационных систем.

Логическая модель представляет собой абстрактное представление данных и может иметь аналоги в реальном мире [1]. Эта модель является универсальной и может использоваться в различных контекстах. На основе логической модели проектировщик может выбрать подходящую СУБД, после чего ERwin автоматически создаст соответствующую физическую модель. На основании физической модели ERwin может сгенерировать соответствующий SQL-скрипт.

3.1 Диаграмма классов

Процесс создания диаграммы классов для web-приложения является важным этапом проектирования, обеспечивающим четкое понимание структуры системы и взаимодействия между ее компонентами. Создание диаграммы классов помогает проектной команде (в случае разрабатываемого приложения – одному разработчику) четко представить архитектуру web-приложения, понять взаимосвязи между его компонентами и обеспечить согласованность при разработке и внедрении системы. Это критически важно для создания устойчивого и масштабируемого web-приложения.

Спроектированная диаграмма классов представлена на рисунке 4.

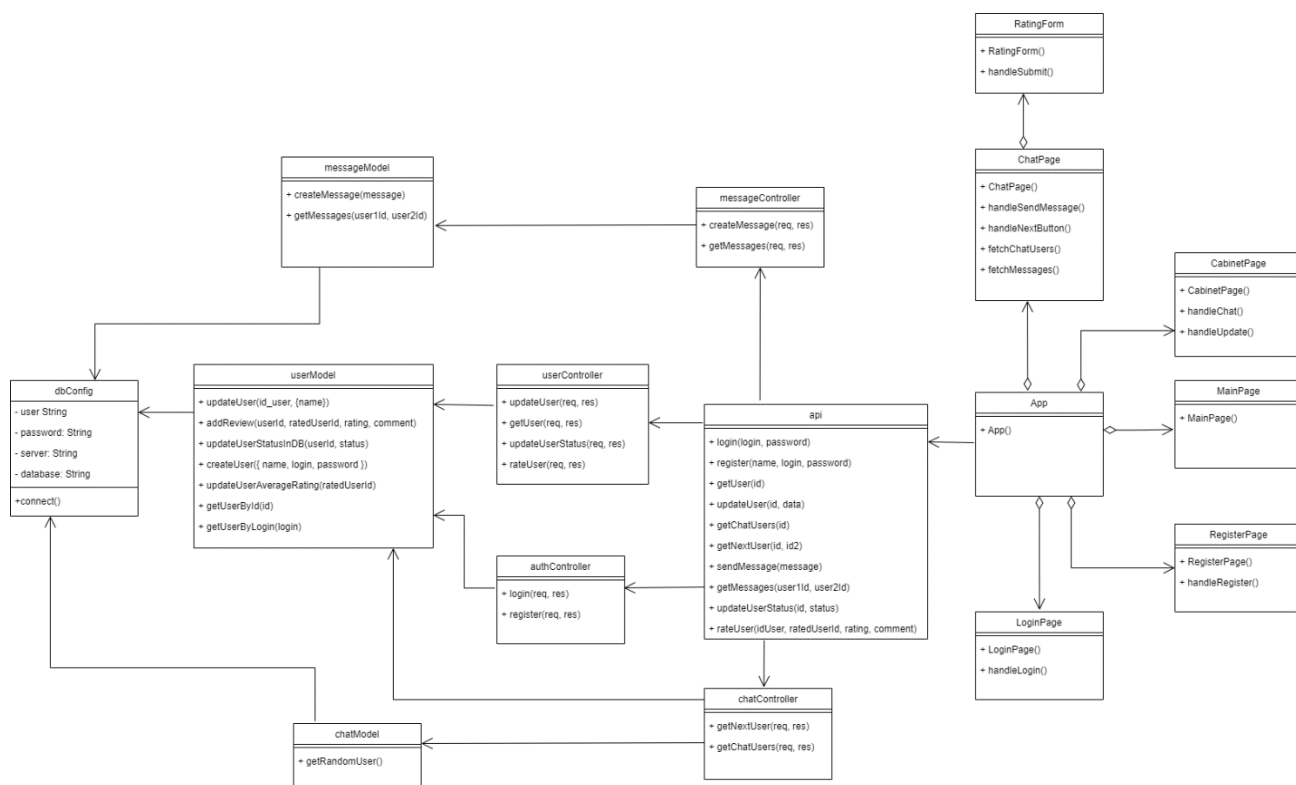


Рисунок 4 – Диаграмма классов

Описание классов и их методов представлены в таблице 2.

Таблица 2 – Описание классов и их методов

Класс	Методы	Описание
1	2	3
App	App()	Вызывает генерацию визуальной части ПИ
ChatPage	ChatPage()	Генерирует страницу чата
	handleSendMessages()	Обработка кнопки «Отправить»
	handleNextButton()	Обработка кнопки «Следующий пользователь»
	fetchChatUsers()	Добавление пользователей в чат
	fetchMessages()	Добавление сообщений в блок чата
RatingForm	RatingForm()	Генерирует форму отзыва о собеседнике
	handleSubmit()	Обработка подтверждения формы
MainPage	MainPage()	Генерирует главную страницу

Продолжение таблицы 2.

1	2	3
CabinetPage	CabinetPage()	Генерирует страницу личного кабинета
	handleChat()	Обработка перехода в чат
	handleUpdate()	Обработка обновления данных о пользователе
RegisterPage	RegisterPage()	Генерирует страницу регистрации
	handleRegister()	Обработка регистрации при нажатии кнопки «Registe»
LoginPage	LoginPage()	Генерирует страницу авторизации
	handleLogin()	Обработка авторизации при нажатии кнопки «Login»
api	login(login, password)	Перенаправляет полученные данные в authController
	Register(name, login, password)	Перенаправляет полученные данные в authController
	getUser(id)	Перенаправляет полученные данные в userController
	updateUser(id,data)	Перенаправляет полученные данные в userController
	getChatUsers(id)	Перенаправляет полученные данные в chatController
	getNextUser(id, id2)	Перенаправляет полученные данные в chatController
	sendMessage(message)	Перенаправляет полученные данные в Controller
	getMessages(user1Id, user2Id)	Перенаправляет полученные данные в messageController
	updateUserStatus(id, status)	Перенаправляет полученные данные в userController
	rateUser(idUser, rateUserId, rating, comment)	Перенаправляет полученные данные в userController

Продолжение таблицы 2.

1	2	3
messageController	createMessage(req, res)	Создаёт запрос с полученными данными к функции в классе messageModel
	getMessages(user1Id, user2Id)	Создаёт запрос с полученными данными к функции в классе messageModel
UserController	updateUser(req, res)	Создаёт запрос с полученными данными к функции в классе userModel
	getUser(req, res)	Создаёт запрос с полученными данными к функции в классе userModel
	updateUserStatus(req, res)	Создаёт запрос с полученными данными к функции в классе userModel
	rateUser(req, res)	Создаёт запрос с полученными данными к функции в классе userModel
chatController	getNextUser(req, res)	Создаёт запрос с полученными данными к функции в классе chatModel
	getChatUsers(req, res)	Создаёт запрос с полученными данными к функциям getUserById(id) в классе userModel и getRandomUser() в классе chatModel
authController	login(req, res)	Создаёт запрос с полученными данными к функции getUserByLogin(login) в классе userModel
	Register(req, res)	Создаёт запрос с полученными данными к функции createUser({name, login, password})

Окончание таблицы 2.

1	2	3
chatModel	getRandomUser()	Обращается к БД с помощью функции connect() из класса dbConfig и получает данные случайного пользователя
userModel	updateUser(id_user, {name})	Обращается к БД с помощью функции connect() из класса dbConfig и обновляет данные пользователя с id = id_user
	addReview(userId, rateUserId, rating, comment)	Обращается к БД с помощью функции connect() из класса dbConfig и добавляет объект в таблицу reviews с полученными данными
	updateUserStatusInDB(userId, status)	Обращается к БД с помощью функции connect() из класса dbConfig и обновляет статус указанного пользователя
	createUser({name, login, password})	Обращается к БД с помощью функции connect() из класса dbConfig и создаёт объект в таблице users
	updateUserAverageRating(ratedUserId)	Обновляет рейтинг пользователя с указанным Id
	getUserById(id)	Возвращает данные пользователя с указанным Id из БД
	getUserByLogin(login)	Возвращает данные пользователя с указанным логином из БД
messageModel	createMessage(message)	Создаёт объект в таблице messages базы данных
	getMessages(user1Id, user2Id)	Возвращает все сообщения указанных пользователей
dbConfig	connect()	Подключается к указанной БД

3.2 Диаграмма IDEF1x

В данном проекте мы сосредоточились на создании именно физической модели данных. Физическая модель позволяет детализировать структуру базы данных, определяя конкретные таблицы, столбцы и их типы данных, а также связи между ними. Это критически важно для реализации базы данных, которая будет эффективно поддерживать требуемые бизнес-процессы и обеспечивать высокую производительность и надежность системы. Спроектированная физическая модель базы данных представлена на рисунке 5.

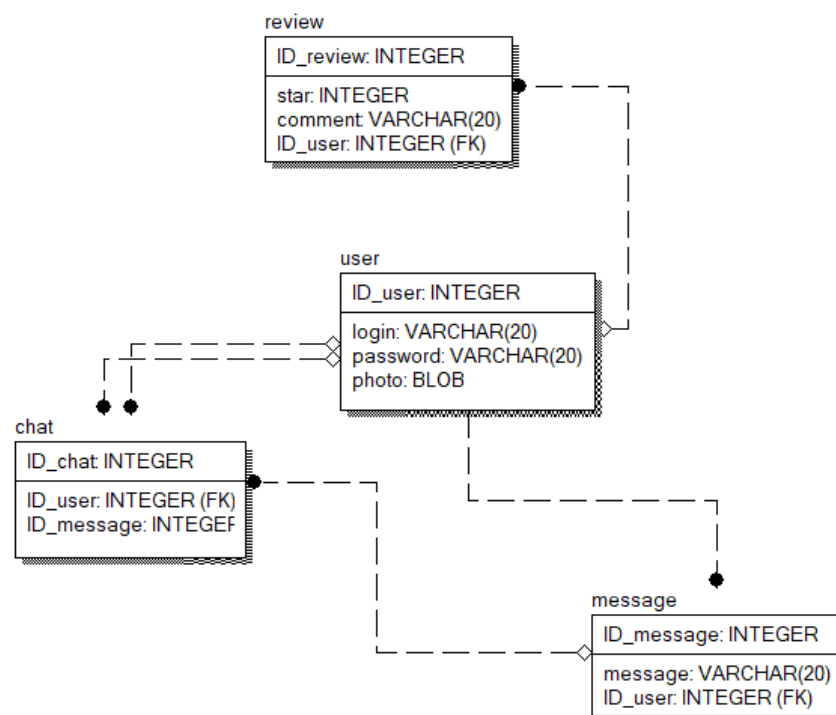


Рисунок 5 – Диаграмма IDEF1x

Представленная диаграмма состоит из сущностей, их атрибутов, и соединений между сущностями.

Описание сущностей и их атрибутов физической модели представлены в таблице 3.

Таблица 3 – Описание сущностей физической модели данных

Сущность	Атрибуты	Размерность	Роль	Описание
1	2	3	4	5
user	ID_user	INTEGER	PK	Уникальный код пользователя
	Login	VARCHAR		Логин пользователя
	Password	VARCHAR		Пароль пользователя
	photo	BLOB		Фото пользователя
chat	ID_chat	INTEGER	PK	Уникальный код чата
	ID_user	INTEGER	FK	Уникальный код пользователя
	ID_message	INTEGER	FK	Уникальный код сообщения
review	ID_review	INTEGER	PK	Уникальный код отзыва
	star	INTEGER		Оценка
	comment	VARCHAR		Комментарий
	ID_user	INTEGER	FK	Уникальный код пользователя
message	ID_message	INTEGER	PK	Уникальный код сообщения
	message	VARCHAR		Тело сообщения
	ID_user	INTEGER	FK	Уникальный код пользователя

4. Проектирование Web-приложения

Проектирование веб-приложений всегда начинается с определения цели проекта. В общем виде цель проекта можно определить как решение ряда взаимосвязанных задач, включающих в себя обеспечение на момент запуска приложения и в течение всего времени его эксплуатации:

- требуемой функциональности приложения и уровня его адаптивности к изменяющимся условиям функционирования;
- требуемой пропускной способности приложения;
- требуемого времени реакции приложения на запросы пользователей;
- безотказной работы приложения;
- необходимого уровня безопасности;
- простоты эксплуатации и поддержки приложения.

Согласно современной методологии, процесс создания веб-приложения представляет собой процесс построения и последовательного преобразования ряда согласованных моделей на всех этапах жизненного цикла (ЖЦ) приложения. На каждом этапе ЖЦ создаются специфичные для него модели - требований к приложению, проекта приложения, требований к функциональным модулям и так далее. Модели формируются рабочими группами команды проекта, сохраняются и накапливаются в репозитории проекта.

Проектирование веб-приложения осуществляется с использованием специальных программных инструментов, таких как UML-диаграммы и CASE-средства. Эти инструменты позволяют создать визуальные представления различных аспектов системы, начиная от диаграмм прецедентов и диаграмм классов до диаграмм последовательности и диаграмм активности. Использование этих инструментов помогает обеспечить согласованность и полноту моделей.

4.1 Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемого приложения доступен каждой группе пользователей.

Для наглядности всех основных функций приложения была составлена диаграмма вариантов использования, представленная на рисунке 6.

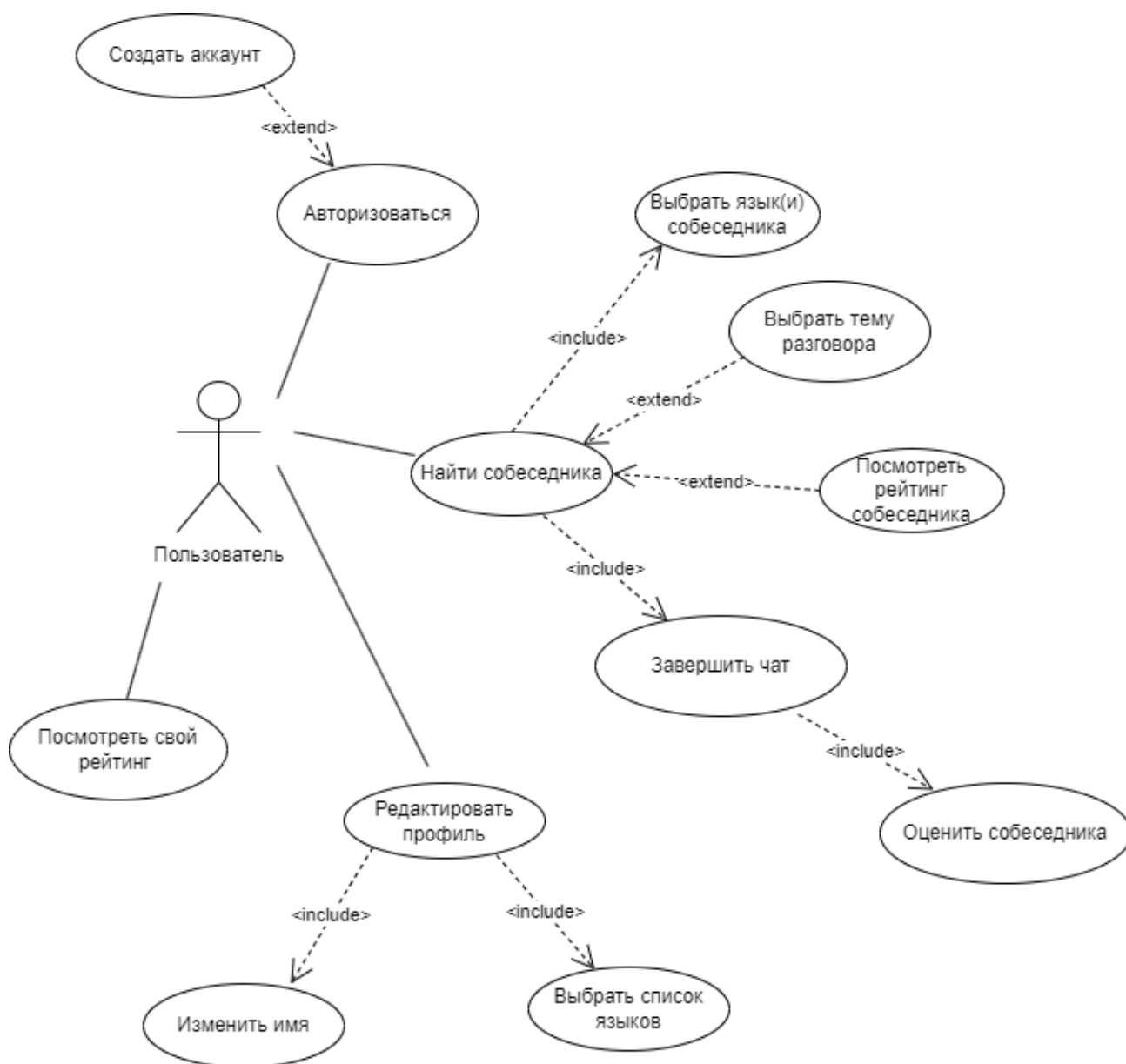


Рисунок 6 – Диаграмма вариантов использования

На диаграмме представлен пользователь, с возможностью посмотреть свой рейтинг, редактировать профиль, а также начать и завершить чат. Для этого сначала пользователю необходимо авторизоваться или зарегистрировать новый

аккаунт. При редактировании профиля пользователь может изменить имя аккаунта, а также выбрать список языков, на которых хочет общаться собеседником. Поиск собеседника после выбора языка, кроме того, пользователь может посмотреть рейтинг собеседника или выбрать тему для разговора. После завершения чата, пользователь должен оценить собеседника.

4.2 Диаграмма деятельности

Суть диаграммы заключается в графическом отображении последовательности действий пользователя и приложения.

Для каждой задачи на диаграмме деятельности указывается, кто выполняет задачу, какие ресурсы необходимы для ее выполнения, какие результаты ожидаются на каждом этапе процесса, а также какая последовательность задач и операций необходима для успешного завершения.

Для представления последовательности выполнения основных задач приложения была составлена диаграмма деятельности.

На представленной диаграмме фигурируют два участника: пользователь и приложение.

Входной точкой процесса является вход пользователя на сайт. После нажатия кнопки «Авторизоваться» система проверит наличие данного аккаунта в БД и при наличии данного аккаунта, произведёт редирект пользователя на главную страницу сайта. Также пользователь может зарегистрировать новый аккаунт и затем авторизоваться под ним. После попадания на главную страницу, у пользователя есть выбор: нажать на кнопку «Мой профиль» или «Начать чат». В первом случае приложение произведёт редирект пользователя на страницу профиля, где профиль можно отредактировать через специальную форму путём нажатия на кнопку «Изменить профиль».

При нажатии на кнопку «Начать чат», открывается форма подбора собеседника. После заполнения формы и нажатия пользователем кнопки «Далее», приложением будет подобран второй пользователь, подходящий под критерии, и оба пользователя будут переведены на страницу совместного чата.

После завершения чата или нажатия на кнопку «Следующий собеседник», приложения попросит пользователя заполнить форму отзыва о собеседнике, и обновит рейтинг собеседника в соответствии с этим отзывом.

Также, после завершения чата, приложение производит перенаправление пользователя на главную страницу.

Диаграмма представлена на рисунке 7.

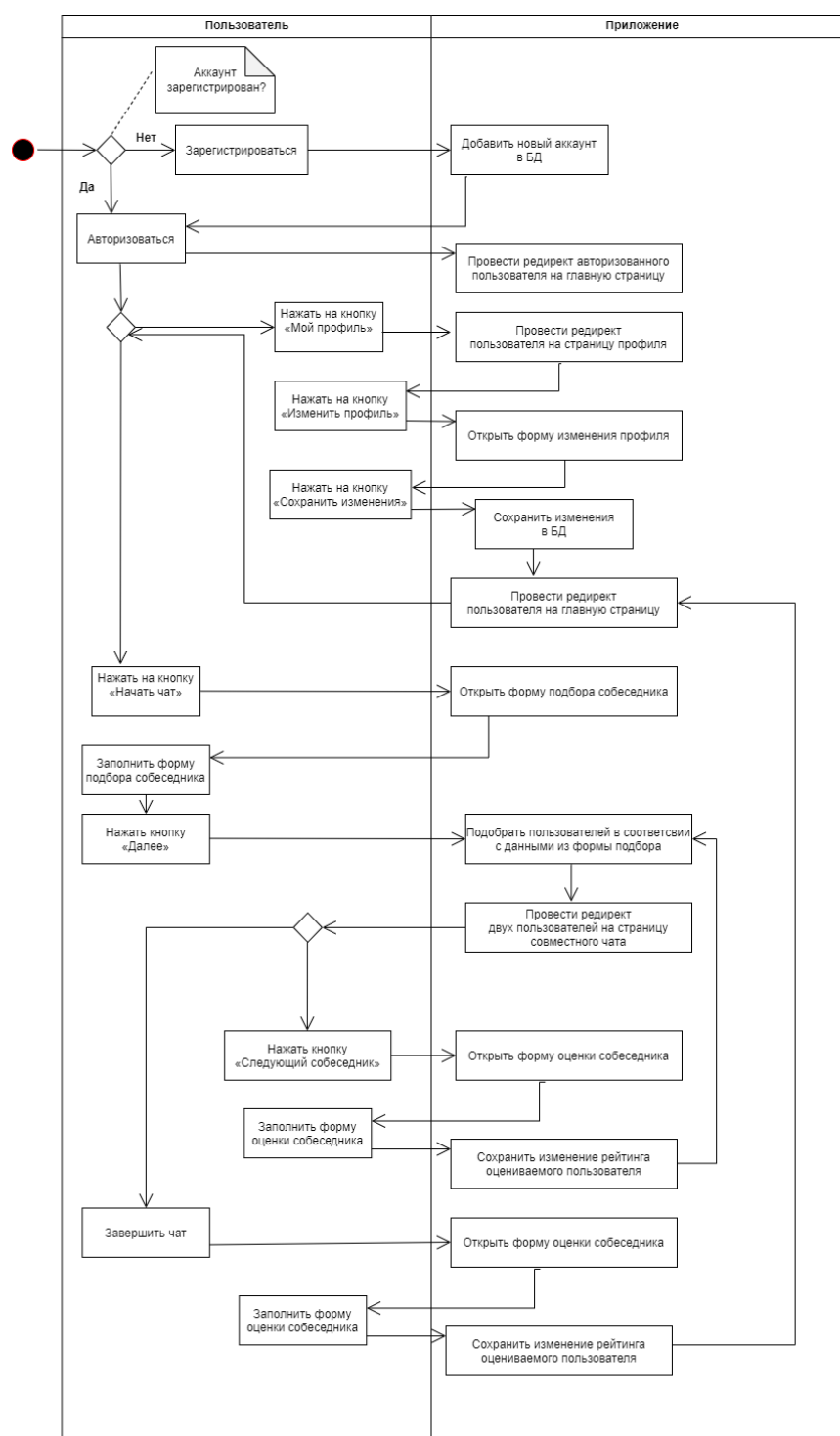


Рисунок 7 – Диаграмма деятельности

4.3 Диаграмма состояний

Диаграмма состояний описывает различные состояния объекта в течение его жизненного цикла и переходы между этими состояниями в ответ на события. Она полезна для моделирования поведения объектов, которые могут находиться в нескольких состояниях.

Основные элементы диаграммы состояний:

- состояния: отражают различные условия объекта в определённый момент времени;
- переходы: Стрелки, соединяющие состояния, показывающие возможные изменения состояния;
- события: Триггеры, которые вызывают переходы между состояниями.

В качестве объекта для разработки диаграммы состояний был взят объект «User» при взаимодействии пользователя с чатом. При разработке были выделены начальное и конечное состояние, а также промежуточные состояния и переходы между ними. Диаграмма состояний представлена на рисунке 8.

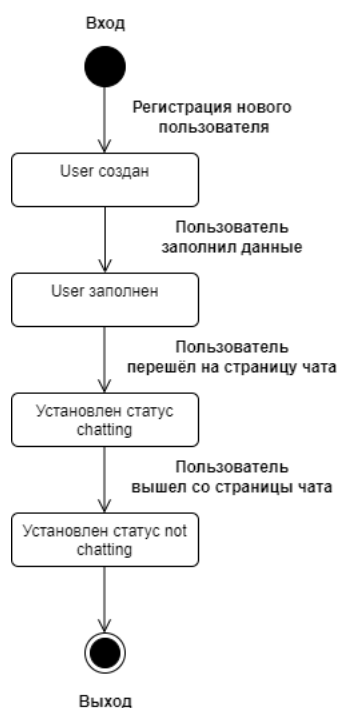


Рисунок 8 – Диаграмма состояний объекта «User»

Спецификации переходов объекта из одного состояния в другое определены в таблице 6.

Таблица 6 – Спецификации переходов объекта user из состояния в состояние

Состояние 1	Событие	Состояние 2	Описание перехода
1	2	3	4
Начальное состояние	Создание объекта user при регистрации	User создан	При регистрации пользователя создаётся незаполненный объект User
User создан	Пользователь заполнил данные	User заполнен	Объект user заполняется данными, введёнными пользователем при регистрации в специальные поля
User заполнен	Пользователь перешёл на страницу чата	Установлен статус «chatting»	Во время перехода пользователя на страницу чата, отправляется запрос в БД, в котором статус объекта user изменяется со стандартного «not chatting», на «chatting», чтобы пользователь смог участвовать в процессе подбора собеседников
Установлен статус «chatting»	Пользователь вышел со страницы чата	Установлен статус «not chatting»	При выходе со страницы чата (закрытие вкладки пользователем или возврат на предыдущую страницу), статус объекта user меняется с «chatting» на «not chatting»
Установлен статус «not chatting»	Установка статуса «not chatting»	Конечное состояние	После установки статуса «not chatting» объект переходит в конечное состояние

Также была создана диаграмма состояний для объекта «Review». Она представлена на рисунке 8, а спецификация переходов данного объекта представлена в таблице 7.

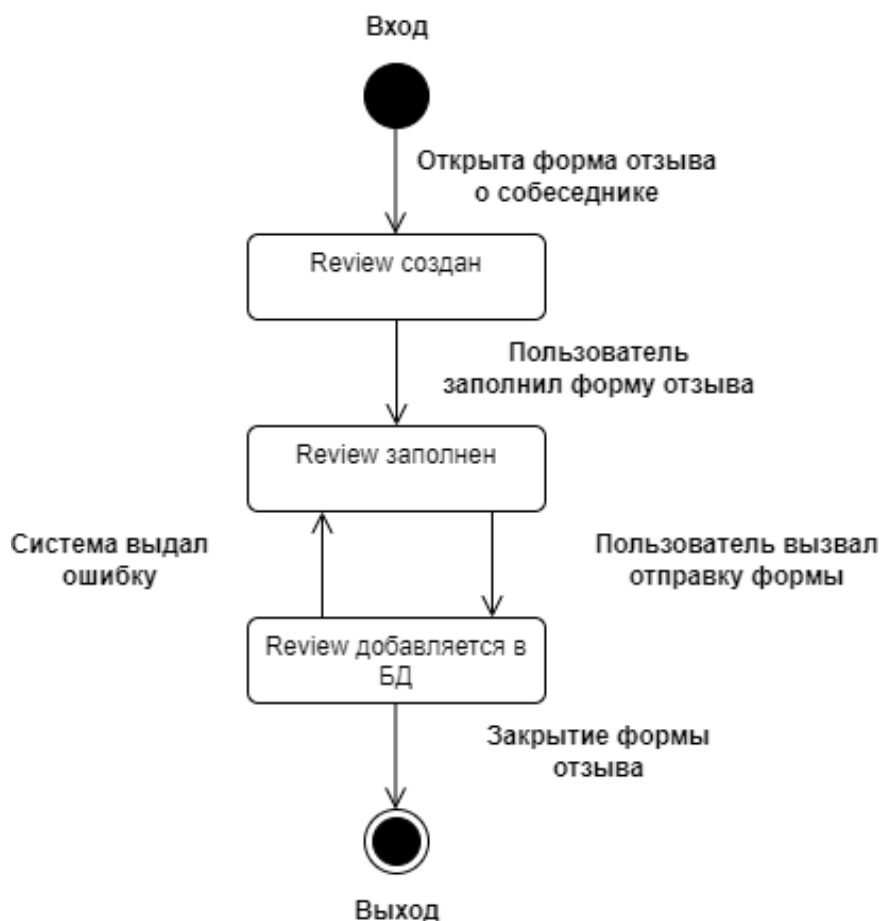


Рисунок 8 – Диаграмма состояний объекта «Review»

Таблица 7 – Спецификации переходов объекта user из состояния в состояние

Состояние 1	Событие	Состояние 2	Описание перехода
1	2	3	4
Начальное состояние	Открыта форма отзыва о собеседнике	Review создан	При переходе к следующему собеседнику, открывается форма для заполнения отзыва о предыдущем собеседнике
Review создан	Пользователь заполнил форму отзыва	Review заполнен	Объект review заполнен данными, введёнными пользователем

Продолжение таблицы 7

1	2	3	4
Review заполнен	Пользователь вызвал отправку формы	Review добавляется в БД	При успешном выполнении запроса, отзыв добавляется в БД
Review добавляется в БД	Система выдала ошибку	Review заполнен	Если запрос не выполнился по каким-либо причинам, объект review не будет добавлен в БД
Review добавляется в БД	Закрытие формы отзыва	Конечное состояние	После успешного выполнения запроса, форма отзыва о собеседнике закрывается, а объект переходит в конечное состояние

5. Проектирование пользовательского интерфейса

Пользовательский интерфейс (ПИ) представляет собой средство взаимодействия между пользователем и системой. Это совокупность всех элементов, через которые пользователь взаимодействует с программным обеспечением или устройством [2]. Пользовательский интерфейс включает в себя графические элементы (иконки, кнопки, окна, формы), звуковые сигналы, тактильные элементы (например, вибрация), а также взаимодействие с пользователем посредством команд и запросов.

Основные критерии оценки пользовательского интерфейса:

1. Удобство использования: Интерфейс должен быть интуитивно понятным и легким в освоении.
2. Эффективность: Он должен обеспечивать быстрый доступ к функциям и информации.
3. Эстетичность: Дизайн интерфейса должен быть привлекательным и не вызывать у пользователя визуального дискомфорта.
4. Последовательность: Единый стиль и поведение элементов интерфейса способствуют лучшему пониманию и предсказуемости.

Пользовательский интерфейс web-приложения представляет собой интерфейс, через который пользователи взаимодействуют с приложением, работающим в веб-браузере. Хорошо спроектированный интерфейс способствует достижению целей пользователя и улучшает общее впечатление от использования приложения.

Пользовательский интерфейс web-приложения состоит из набора страниц и элементов на этих страницах. Для данного приложения было решено разработать следующие страницы:

- начальная страница;
- страница авторизации;
- страница регистрации;
- личный кабинет;

– страница чата.

Начальная страница – это страница, на которую по умолчанию попадают все пользователи, перешедшие по адресу сайта. Она содержит краткое описание приложения, а также элементы управления, которые перенаправляют на страницу авторизации или регистрации. Изображение начальной страницы представлено на рисунке 9.

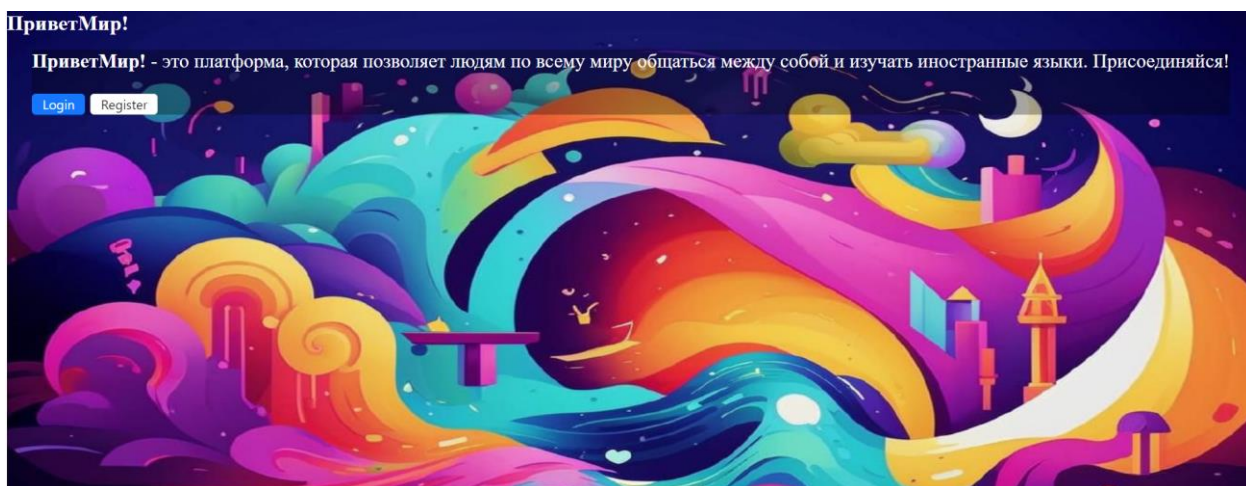


Рисунок 9 – Начальная страница

Кнопка «Login» при нажатии перенаправляет пользователя на страницу авторизации, а кнопка «Register» соответственно на страницу регистрации.

Страница авторизации – это страница, на которой проходит процесс авторизации зарегистрированного пользователя.

Для успешного выполнения данного процесса, пользователь должен ввести логин и пароль, соответствующие данным из базы данных приложения, в предназначенные для этого поля, а затем нажать кнопку авторизации.

После успешной авторизации, происходит перенаправление пользователя на страницу личного кабинета, при неуспешной авторизации, пользователь остаётся на той же странице.

Страница авторизации представлена на рисунке 10.



Рисунок 10 – Страница авторизации

В поля, подписанные как «Логин» и «Пароль» вводятся данные пользователя, кнопка «Login» запускает процесс проверки введённых данных.

Страница регистрации – это страница, на которой пользователь может создать новый аккаунт. Для этого ему нужно ввести новый логин и пароль, а также задать имя, под которым его будут видеть остальные пользователи. Страница регистрации представлена на рисунке 11.

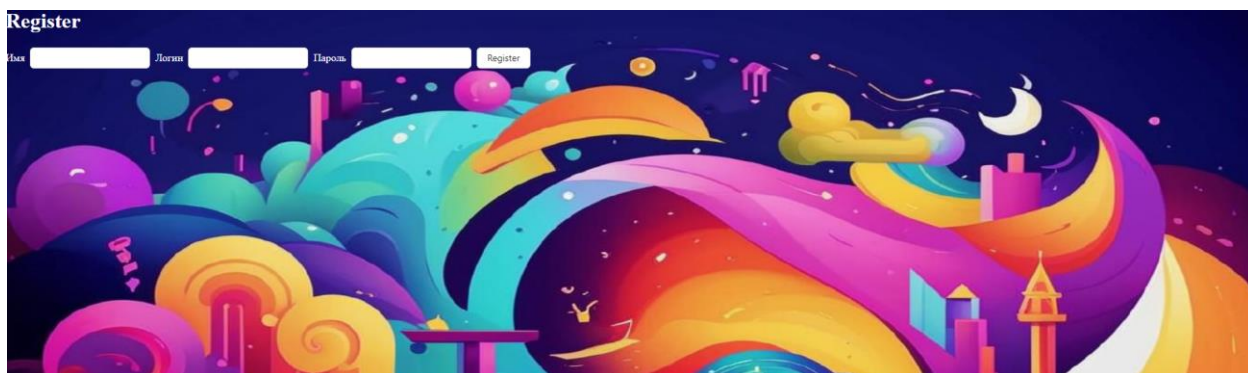


Рисунок 11 – Страница регистрации

В поля, подписанные как «Имя», «Логин» и «Пароль» вводятся данные пользователя, кнопка «Register» запускает процесс занесения введённых данных в базу данных приложения. После успешной регистрации, пользователь перенаправляется на страницу авторизации, где может войти по введённым ранее данным.

Личный кабинет – это страница, на которой пользователь может увидеть свои данные (Отображаемое имя, рейтинг, изучаемые и знакомые языки), а также изменить их. Также с этой страницы пользователь может перейти на страницу чата. Страница личного кабинета представлена на рисунке 12.

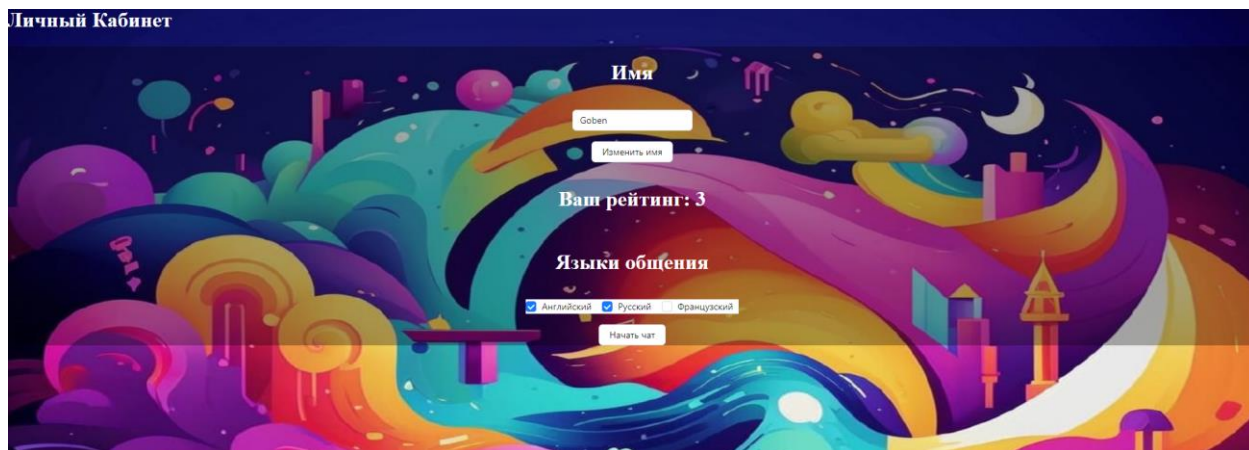


Рисунок 12 – Личный кабинет

В поле «Имя» отображается имя аккаунта таким, как его видят остальные пользователи. В поле «Ваш рейтинг» отображается рейтинг пользователя, составленный на основе оценки другими пользователями. Рейтинг может принимать значение от 1 до 5 или 0, если оценки отсутствуют. В панели «Языки общения» пользователь может выбрать один или несколько языков путём выставления «галочек». Собеседники в чате будут подбираться в соответствии с выбранными языками. Кнопка «Начать чат» перенаправляет пользователя на страницу чата.

Страница чата – это основная функциональная страница данного web-приложения. При переходе на неё пользователю подбирается собеседник в соответствии с настройками, сделанными пользователем в личном кабинете. Страница разделена на несколько основных блоков:

- данные пользователей;
- видеоизображение собеседника;
- список сообщений;
- управляющие элементы.

Страница чата представлена на рисунке 13.

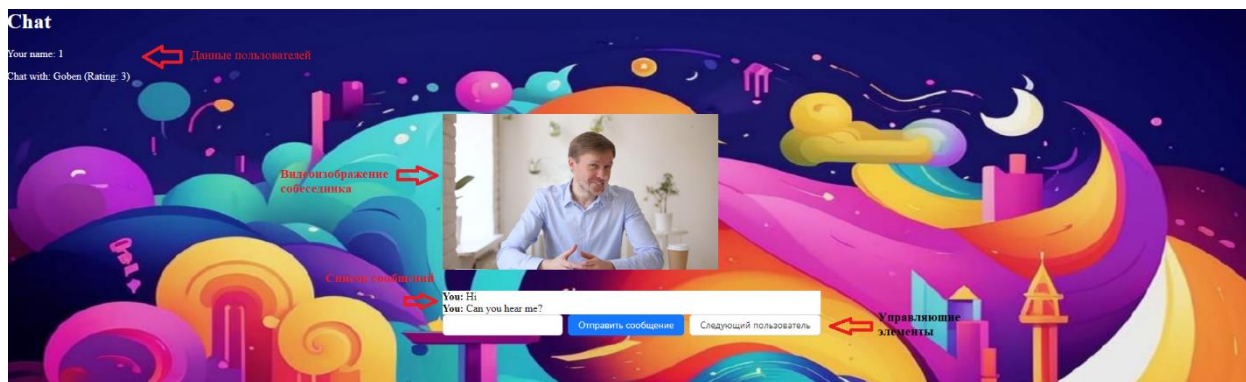


Рисунок 13 – Страница чата

В блоке «Данные пользователей» отображается ваше имя, а также имя собеседника и его рейтинг. Блок «Видеоизображение собеседника» передаёт изображение с веб-камеры собеседника.

В блоке «Список сообщений» содержатся все сообщения, написанные в данном конкретном чате, а также подписывается отправитель «You» в случае, если сообщение отправил пользователь или имя собеседника, если сообщение было отправлено им.

Блок «Управляющие элементы» содержит поле для написания сообщения, кнопку «Отправить сообщение», которая осуществляет отправку написанного сообщения, а также кнопку «Следующий пользователь», которая отвечает за процесс смены собеседника.

В процессе смены собеседника, пользователю будет предложено оценить (по шкале от 1 до 5) опыт общения с предыдущим собеседником, также пользователь имеет возможность оставить комментарий. Рейтинг оцениваемого собеседника будет изменён в соответствии с выставленной оценкой.

Форма оценки собеседника отображается на странице чата, а не по отдельному адресу.

Изображение формы оценки собеседника представлено на рисунке 14.

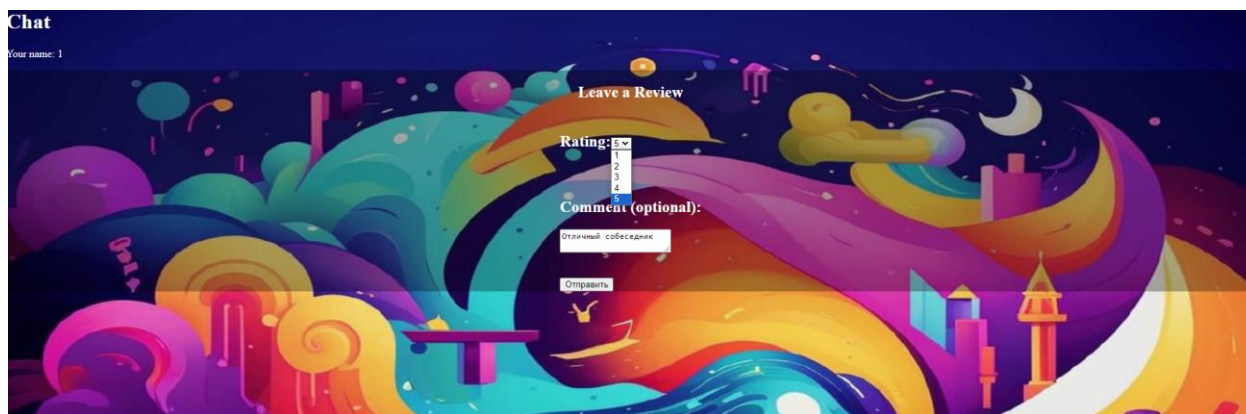


Рисунок 14 – Форма оценки собеседника

Пользователь может выставить оценку, выбрав её из выпадающего списка и написать комментарий о своём опыте общения. При нажатии на кнопку «Отправить», результат будет записан в базу данных приложения, а пользователь будет перенаправлен обратно на страницу чата, но уже с другим собеседником.

6. Проведение функционально-стоимостного анализа проекта

Функционально-стоимостный анализ (ФСА) – метод системного исследования функций отдельного изделия или технологического, производственного процесса, организационной структуры, ориентированный на повышение эффективности использования ресурсов путем оптимизации соотношения между потребительскими свойствами объекта исследования и затратами на его разработку, производство и эксплуатацию.

Цель ФСА состоит в достижении предприятием максимальной реализации поставленных целей при рациональном использовании ресурсов [3].

В качестве основных моментов при разработке системы рассматриваются: оплата труда программиста и стоимость электроэнергии.

Также стоит учитывать, что сложность и объем разработки зависят от определенной среды и сроков выполнения работы.

Основные компоненты затрат на разработку Web-приложения показаны на рисунке 15.

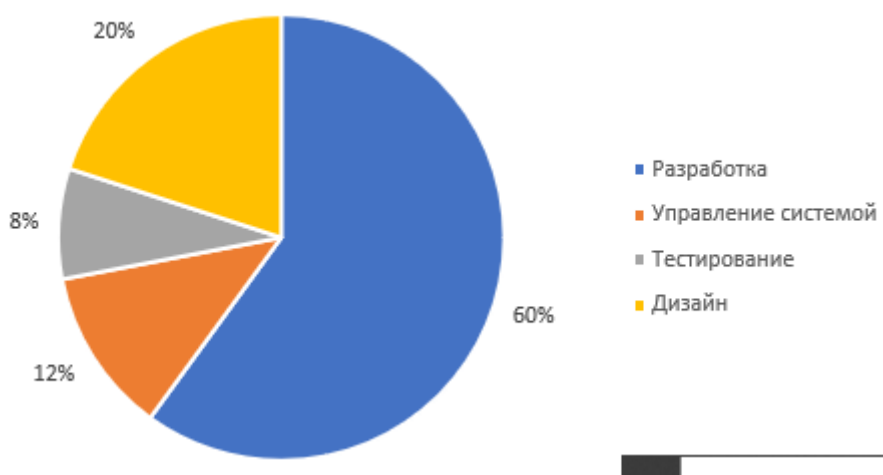


Рисунок 15 – Основные компоненты затрат на разработку

По результатам исследования сервиса «Хабр Карьера», в первом полугодии 2024 г. размер оплаты труда IT-специалистов вырос до ₽167,5 тыс.

При рабочей неделе в 40 часов, в месяц выходит 160 часов работы.

Это означает, что разработчик в среднем получает $167500/160=1046$ руб/час. Компьютер разработчика расходует приблизительно 0,6 кВт электроэнергии. Тариф на электроэнергию составляет 3,63 рубля за кВт.

Выполнение выпускной квалификационной работы (ВКР) состоит из пяти бизнес-процессов: «Анализ предметной области», «Анализ требований и проектирование», «Реализация», «Тестирование» и «Оформление ВКР».

Был проведен анализ затрат времени и количества дней выполнения на каждый этап для одного разработчика.

Для наглядного графического отображения плана разработки ВКР была спроектирована диаграмма Ганта, представленная на рисунке 16.

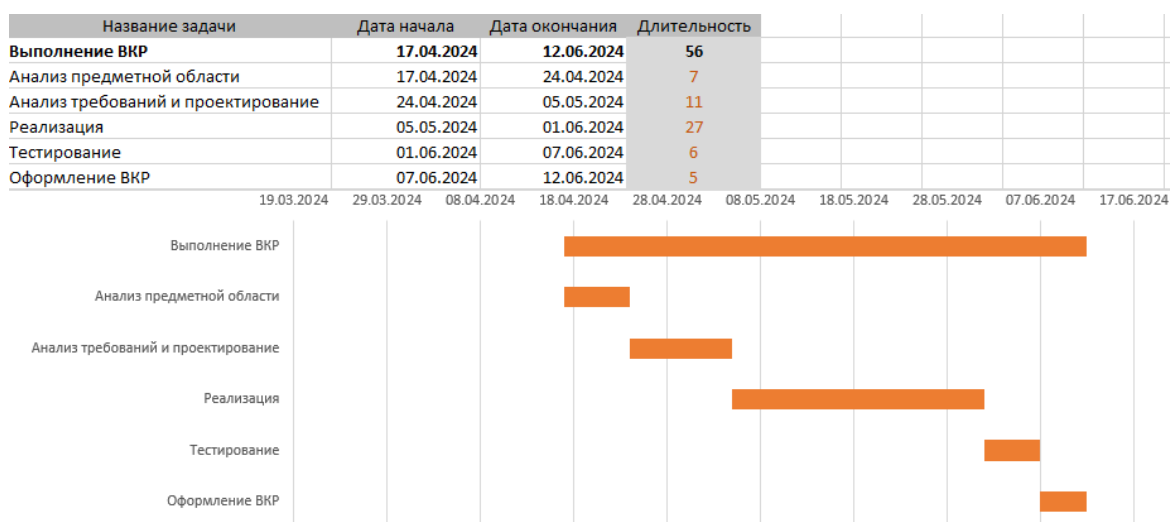


Рисунок 16 – Диаграмма Ганта

Учитывая все факторы, были сделаны вычисления, которые представлены в таблице 8.

Таблица 8 – Затраты на разработку

Этапы	Время на выполнение (часы)	Тариф (рублей в час)	Затраты на электроэнергию (рублей в час)	Всего
Анализ предметной области	7	1046	2	7336
Анализ требований и проектирование	11	1046	2	11528
Реализация	27	1046	2	28296
Тестирование	6	1046	2	6288
Оформление ВКР	5	1046	2	5240
Всего	56			58688

Предположительная стоимость разработки продукта составляет 58688 рублей.

7. Программная реализация

Проект реализован на основе клиент-серверной архитектуры. Серверная часть написана на языке JavaScript с использованием платформы Node.js и фреймворка Express. Клиентская часть разработана с использованием React. Взаимодействие между клиентом и сервером осуществляется через REST API.

Для удобства, рассмотрим сначала реализацию архитектуры данного приложения, а затем реализацию кода в отдельных компонентах.

7.1 Реализация архитектуры

Продемонстрировать компоненты, из которых состоит разрабатываемое приложение можно с помощью диаграммы компонентов, построенной с помощью UML-средств [7].

Диаграмма компонентов описывает физические части системы и их отношения [6]. Компоненты могут быть программными модулями, библиотеками, внешними системами и так далее. Эта диаграмма помогает понять, как компоненты взаимодействуют друг с другом в системе.

Для большей наглядности было решено отобразить архитектуру приложения на 3-х различных диаграммах компонентов:

- обобщенная диаграмма компонентов;
- диаграмма компонентов пользовательского интерфейса (ПИ);
- диаграмма компонентов серверной части.

Обобщенная диаграмма компонентов помогает понять, как взаимодействуют крупные компоненты приложения: база данных (languageNew), серверная часть (backend) и пользовательский интерфейс (frontend). Данная диаграмма представлена на рисунке 17.

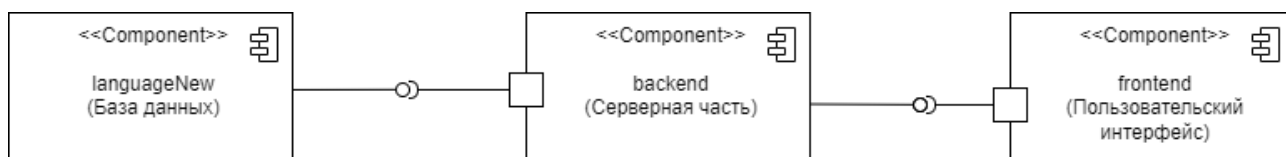


Рисунок 17 – Обобщенная диаграмма компонентов

На данной диаграмме видно, что серверная часть представляет интерфейс взаимодействия с базой данных приложения, а пользовательский интерфейс в свою очередь представляет понятный пользователю интерфейс взаимодействия с серверной частью.

В компоненте «frontend» (ПИ) находятся те компоненты, которые видны пользователю при использовании Web-приложению, а также компоненты для связи с серверной частью. Диаграмма компонентов пользовательского интерфейса представлена на рисунке 18.

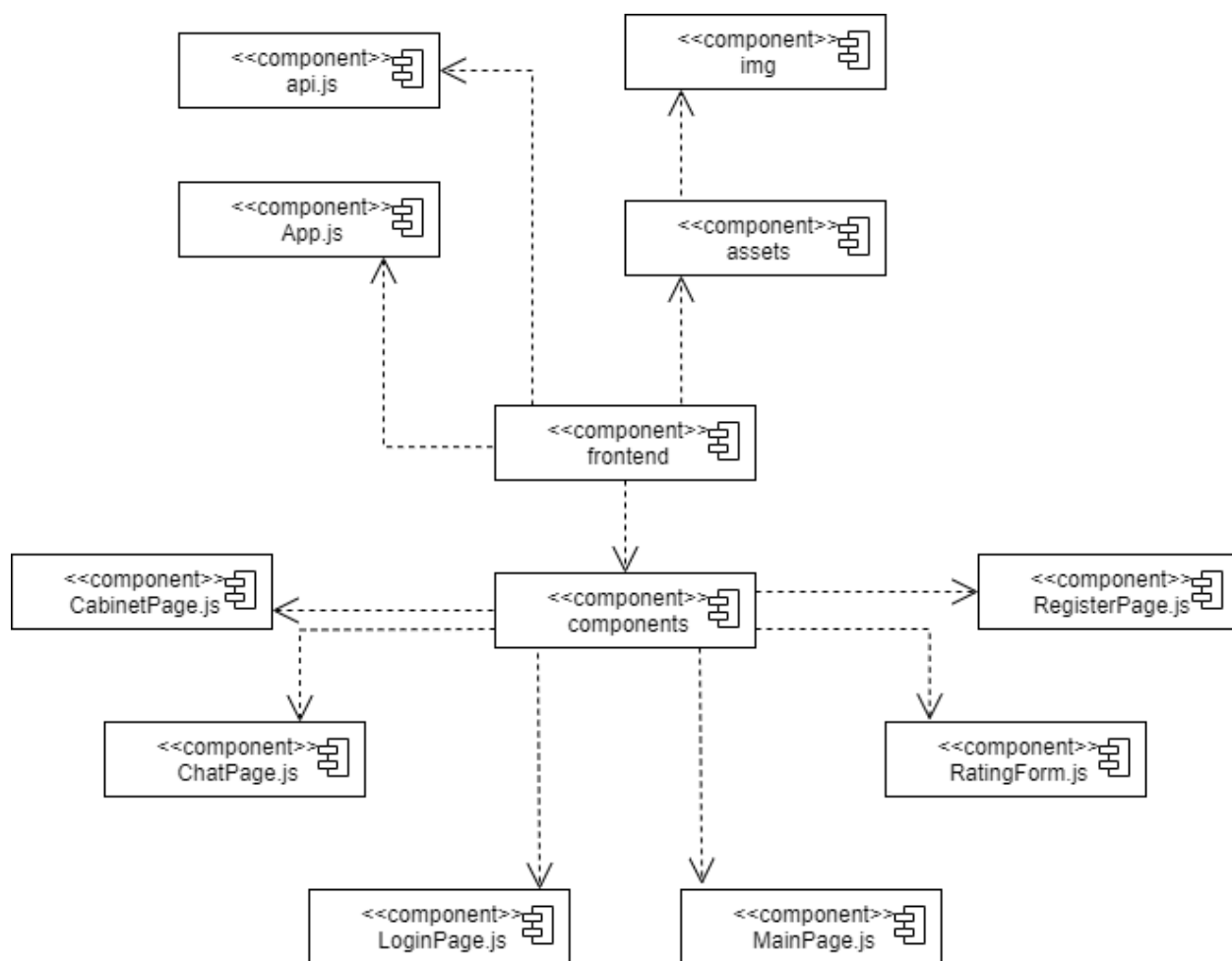


Рисунок 18 – Диаграмма компонентов ПИ

Рассмотрим подробнее компоненты данной диаграммы:

1. Компонент «frontend» включает в себя: «App.js», «api.js», «assets», «components»;
2. Компонент «assets» является хранилищем медиа-файлов и включает в себя компонент «img»;

3. Компонент «App.js» является тем компонентом, в котором содержатся ссылки на остальные компоненты ПИ;
4. Компонент «components» объединяет все модули страниц приложения и включает в себя: «CabinetPage.js», «ChatPage.js», «LoginPage.js», «MainPage.js», «RatingForm.js», «RegisterPage.js»;
5. Компонент «api.js» является портом для взаимодействия с серверной частью.;
6. Компонент «img» содержит все изображения, используемые в интерфейсе приложения, например изображение фона «background.jpg»;
7. Компонент «CabinetPage.js» является модулем для генерации страницы личного кабинета;
8. Компонент «ChatPage.js» является модулем для генерации страницы чата;
9. Компонент «LoginPage.js» является модулем для генерации страницы авторизации;
10. Компонент «MainPage.js» является модулем для генерации страницы главной страницы;
11. Компонент «RatingForm.js» является модулем для генерации формы отзыва о собеседнике;
12. Компонент «RegisterPage.js» является модулем для генерации страницы регистрации.

В компоненте «backend» находятся модули, обрабатывающие запросы, поступающие из ПИ, а также компоненты для взаимодействия с базой данных приложения. Диаграмма компонентов серверной части представлена на рисунке 19.

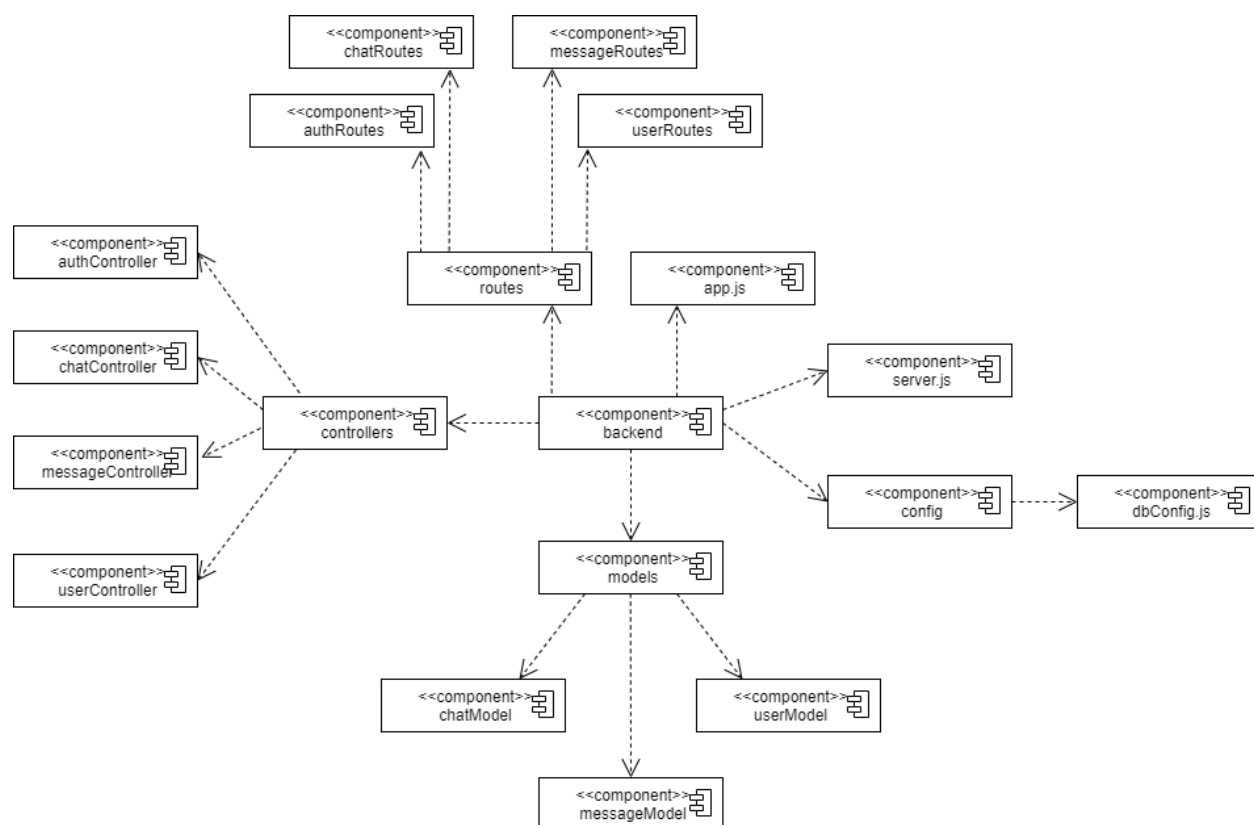


Рисунок 19 – Диаграмма компонентов серверной части

Рассмотрим подробнее компоненты данной диаграммы:

1. Компонент «backend» включает в себя: «routes», «app.js», «server.js», «config», «controllers» и «models»;
2. Компонент «routes» определяет маршруты для обработки запросов к серверу. В этом модуле описаны пути и соответствующие контроллеры для различных операций, таких как регистрация пользователей, аутентификация, управление чатами и отзывами. Он включает в себя компоненты «authRoutes», «chatRoutes», «messageRoutes», «userRoutes»;
3. Компонент «controllers» содержит логику обработки запросов и взаимодействия с моделями базы данных. Контроллеры обрабатывают входящие запросы, выполняют бизнес-логику и возвращают ответы клиенту. Он включает в себя компоненты «authController», «chatController», «messageController» и «userController»;
4. Компонент «app.js» является основным приложением Express. В этом файле происходит настройка серверного приложения, включая

подключение всех маршрутов и настройка CORS Cross-Origin Resource Sharing) для разрешения запросов с фронтенда;

5. Компонент «server.js» отвечает за запуск сервера. Он инициализирует приложение Express из `app.js` и запускает сервер на определенном порту;

6. Компонент «config» содержит конфигурационные файлы, используемые в приложении. Он включает в себя компонент «dbConfig.js»;

7. Компонент «models» определяет схемы данных и взаимодействие с базой данных. Модели описывают методы для выполнения CRUD (Create, Read, Update, Delete) операций. Он содержит в себе компоненты «userModel.js», «messageModel.js», «chatModel.js»;

8. Компонент «userController.js» отвечает за обработку запросов, связанных с пользователями, таких как регистрация, аутентификация и обновление профиля;

9. Компонент «chatController.js» отвечает за обработку запросов, связанных с чатом, например добавление пользователей в чат;

10. Компонент «messageController.js» отвечает за обработку запросов, связанных с сообщениями, таких как отправка сообщений;

11. Компонент «authController.js» отвечает за обработку запросов, связанных с регистрацией;

12. Компонент «authRoutes.js» содержит маршруты, связанные с процессами регистрации и авторизации;

13. Компонент «userRoutes.js» содержит маршруты, связанные с пользователями;

14. Компонент «chatRoutes.js» содержит маршруты, связанные с чатами

15. Компонент «messageRoutes.js» содержит маршруты, связанные с сообщениями;

16. Компонент «userModel.js» содержит методы для работы с данными пользователей;

17. Компонент «chatModel.js» содержит методы для работы с данными чатов;
18. Компонент «messageModel.js» содержит методы для работы с данными сообщений;
19. Компонент «dbConfig.js» содержит настройки для подключения к базе данных приложения «languageNew».

Также для данного приложения была создана диаграмма развёртывания. Диаграмма развёртывания (Deployment Diagram) показывает физическое расположение программных компонентов на различных узлах (например, серверах, устройствах) [5]. Она полезна для понимания, как программное обеспечение развёрнуто в реальной среде, и как различные узлы взаимодействуют друг с другом.

Диаграмма развёртывания включает в себя следующие элементы:

- узлы – визуально отображаются в виде куба, представляет собой физическую сущность, которая выполняет одну или несколько компонентов, подсистем или исполняемых файлов. Узел может быть аппаратным или программным элементом;
- артефакты – это конкретные элементы, которые вызваны процессом разработки. Примерами артефактов являются библиотеки, архивы, конфигурационные файлы, исполняемые файлы и т.д.;
- коммуникационная ассоциация – визуально отображается в виде линии между двумя узлами. Она показывает путь связи между ними.

Диаграмма развёртывания для разрабатываемого Web-приложения представлена на рисунке 20.

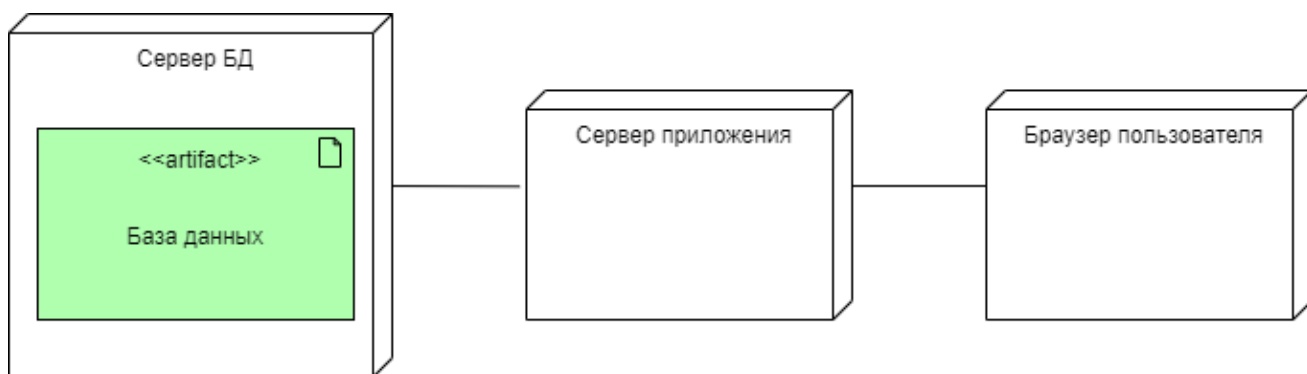


Рисунок 20 – Диаграмма развертывания

На представленной диаграмме в качестве узлов выступают следующие элементы:

1. Сервер базы данных (Сервер БД). В данном приложении был использован SQL Server под управлением SQL Server Management Studio 2020;
2. Сервер приложения. Был развернут локальный сервер под управлением Node.js на порте :5000;
3. Браузер пользователя – это браузер, на котором можно открыть разрабатываемое web-приложение.

В качестве браузера пользователя может выступать любой браузер с поддержкой JavaScript, в данном случае при тестировании web-платформы будут использованы браузеры «Google Chrome», «Mozilla Firefox» и «Яндекс Браузер».

Артефактом выступает созданная база данных «languageNew». БД располагается на локальном SQL-сервере, под управление MSSQL MS. Она представлена на рисунке 21.

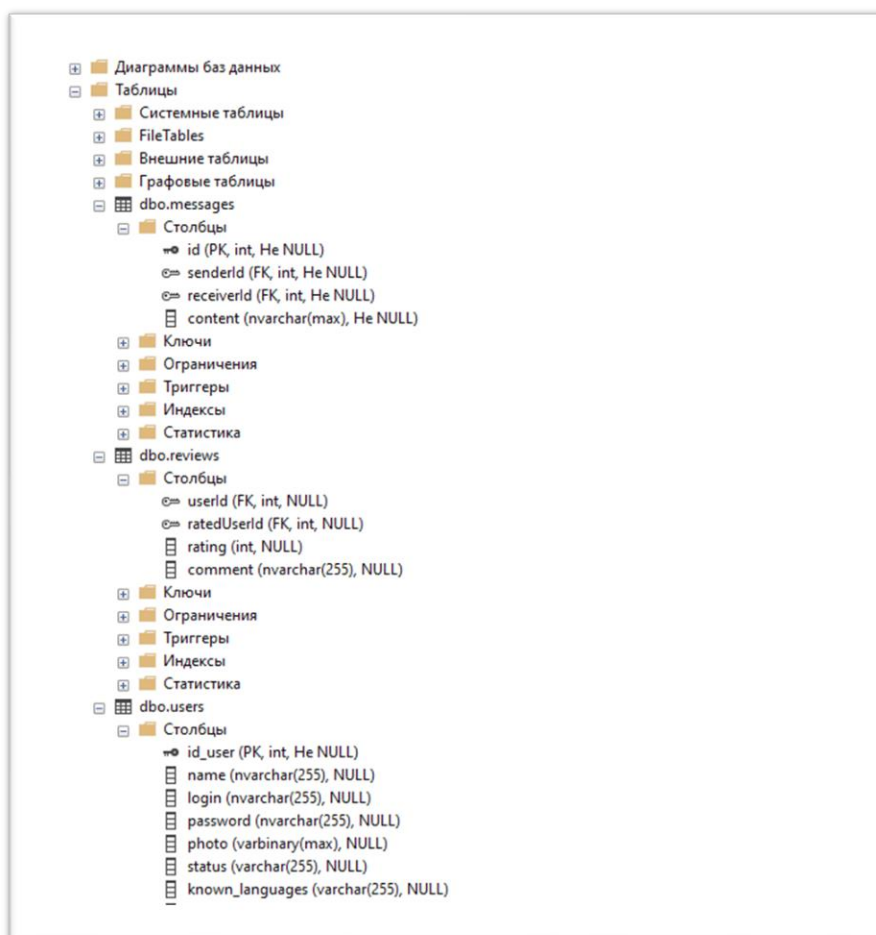


Рисунок 21 – База данных «languageNew» в среде MSSQL MS

7.2 Реализация кода Web-приложения

Все модули разрабатываемого Web-приложения можно условно разделить на 3 типа [11]:

- модули генерации визуальных компонентов;
- модули обработки запросов;
- модули взаимодействия с базой данных.

Приведём наглядный пример модуля генерации визуальных компонентов. Для этого обратимся к модулю «MainPage.js», который генерирует главную страницу приложения. Код данного модуля приведён в листинге 1.

Листинг 1 – Программный код модуля «MainPage.js»

```
import React from 'react';
import { Link } from 'react-router-dom';
import { Button, Space, Flex } from 'antd';
```

```

const MainPage = () => {
  return (
    <div style={{backgroundImage:
`url(${require('../assets/img/background.jpg')})`, backgroundSize: '100%
100%', height: '90vh', width: '100%'}}>
      <h1 style={{color: "white"}}>ПриветМир!</h1>
      <Flex gap='middle' align='center' justify='center' vertical>
        <text style={{
          color: "white", fontSize: 30, backgroundColor: 'rgb(0 0 0
/ 30%)' }}> <b> ПриветМир! </b>
          - это платформа, которая позволяет людям по всему миру
общаться между собой и изучать иностранные языки.
          Присоединяйся!
        <p/>
        <Space>
        <Button type='primary'>
          <Link to="/login" style={{fontSize: 20, textAlign:
"center"}}>
            Login
          </Link>
        </Button>
        <Button>
          <Link to="/register" style={{fontSize: 20, textAlign:
"center"}}>
            Register
          </Link>
        </Button>
        </Space>
      </text>
    </Flex>
  </div>
);
};

export default MainPage;

```

В первом блоке мы видим «импорты» - это используемые модулем компоненты из других модулей или библиотек. В данном случае помимо стандартного для React импорта «React from 'react'», мы можем увидеть импорт компонента «Link» для создания ссылок в тексте и нескольких компонентов из библиотеки «antd», используемых для генерации некоторых визуальных компонентов.

Далее идёт объявление основной функции модуля «MainPage», которая возвращает набор созданных компонентов, внутри основного компонента <div>. Также мы видим использование атрибута «style={{}}» при создании некоторых компонентов. Этот атрибут позволяет редактировать стиль элемента аналогично использованию атрибута «style» в HTML.

Блок «export» встраивает модуль в структуру проекта и позволяет остальным модулям подключать его.

Далее рассмотрим модуль обработки запросов на примере модуля «authController.js». Код данного модуля представлен в листинге 2.

Листинг 2 – Программный код модуля «authController.js»

```
const userModel = require('../models/userModel');

const login = async (req, res) => {
  const { login, password } = req.body;
  const user = await userModel.getUserByLogin(login);
  if (user && user.password === password) {
    res.json({ success: true, userId: user.id_user });
  } else {
    res.json({ success: false, message: 'Invalid login or password',
login: login, password: password, user: user});
  }
};

const register = async (req, res) => {
  const { name, login, password } = req.body;
  const result = await userModel.createUser({ name, login, password });
  if (result) {
    res.json({ success: true });
  } else {
    res.json({ success: false, message: 'Registration failed' });
  }
};

module.exports = {
  login,
  register
};
```

В начальном блоке модуля происходит подключения модуля «userModel» через стандартную Node.js функцию «require». Затем идёт создание асинхронной (async) функции «login», в которой Извлекаются login и password из тела запроса (req.body), затем происходит вызова функции «getUserByLogin(login)» из модуля «userModel» для поиска пользователя с необходимым логином, затем сравнивается пароль из тела запроса и найденный пароль из БД. По итогам сверки возвращается успешный или неуспешный результат в зависимости от совпадения паролей.

Функция «Register» берёт данные из тела запроса и создаёт нового пользователя с помощью функции «createUser» из модуля «userModel».

По итогам выполнения вызванной функции так же возвращается успешный (success: true) или неуспешный (success: false) результат.

В последнем блоке (module.exports) происходит экспорт описанных функций, чтобы другие модули могли их использовать.

Модули взаимодействия с базой данных располагаются в компоненте «models». Рассмотрим один из этих модулей: «chatModel.js». Код данного модуля представлен в листинге 3.

Листинг 3 – Программный код модуля «chatModel.js»

```
const sql = require('mssql');
const db = require('../config/dbConfig');

const getRandomUser = async () => {
  const pool = await db.connect();
  const result = await pool.request()
    .query('SELECT TOP 1 * FROM [users] WHERE status = \'chatting\'
ORDER BY NEWID() ');
  return result.recordset[0];
};

module.exports = {
  getRandomUser,
};
```

В начальном блоке происходит подключение модулей «mssql» и «dbConfig», которые обеспечивают возможность отправлять SQL запросы к базе данных приложения и обрабатывать полученный результат.

Далее объявляется асинхронная функция «getRandomUser», которая подбирает случайного пользователя из БД, у которого присутствует статус «chatting». Эта функция используется для подбора собеседников в чатах.

Функция подключается к базе данных при помощи функции «connect()», а затем отправляет ей SQL запрос (request().query()), результат выполнения которого записывается в константу «result». Затем в последнем блоке модуля происходит экспорт функции «getRandomUser».

Программный код остальных модулей Web-приложения представлен в приложении А.

8. Тестирование Web-приложения

Тестирование программного обеспечения является неотъемлемой частью процесса разработки, обеспечивая качество, надежность и соответствие продукта требованиям. Существуют различные виды тестирования, каждый из которых решает специфические задачи и используется на разных этапах жизненного цикла разработки. Ниже представлены основные виды тестирования и их особенности:

- модульное тестирование – предполагает проверку отдельных модулей или компонентов программы в изоляции от других частей системы;
- интеграционное тестирование – направлено на проверку взаимодействия между различными модулями или компонентами системы;
- нагрузочное тестирование – проверяет систему под различными уровнями нагрузки для оценки её производительности;
- тестирование безопасности – направлено на выявление уязвимостей в системе, которые могут быть использованы злоумышленниками;
- тестирование совместимости – проверяет, как система работает в различных средах (например, браузерах);
- функциональное тестирование - проверяет, что конкретные функции приложения работают в соответствии с требованиями;
- тестирование пользовательского интерфейса – помогает выявить и исправить дефекты интерфейса.

8.1 Выбор и реализация метода тестирования

Основная задача разрабатываемого приложения – это предоставлять удобный и функциональный интерфейс для взаимодействия пользователя друг с другом. Для осуществления поставленной задачи от приложения требуется:

- исправность всех используемых функций;
- наличие кроссплатформенности (приложение должно открываться во всех популярных браузерах);

- интуитивно понятный и визуально приятный пользовательский интерфейс.

Чтобы удостовериться, что приложение соответствует всем названным выше требованиям, было проведено автоматизированное функциональное тестирование, тестирование совместимости для браузеров «Google Chrome», «Mozilla Firefox» и «Яндекс Браузер», а также было осуществлено тестирование пользовательского интерфейса.

8.2 Функциональное тестирование

Автоматизированное функционально тестирование было произведено при помощи фреймворка Playwright [10]. Playwright – это фреймворк с открытым исходным кодом для автоматизации тестирования конечного пользователя. Он поддерживает большинство современных браузеров, включая Firefox, Chromium и Webkit с использованием единого API. Playwright создан и поддерживается компанией Microsoft и постепенно завоевывает популярность.

При выборе инструмента тестирования также рассматривался фреймворк «WebdriverIO», но при сравнении данных фреймворков, Playwright показал своё преимущество [12]. При сравнении WebdriverIO и Playwright ключевыми факторами стали скорость Playwright, поддержка записи видео и поддерживаемые языки программирования.

Кроме того, анализ трендов установки через Npm показывает, что количество установок Playwright недавно превзошло количество установок WebdriverIO [4]. Этот тренд указывает на то, что Playwright, вероятно, станет будущим лидером среди фреймворков для многобраузерного e2e тестирования веб-приложений, что, в свою очередь, приведет к росту числа разработчиков и поддержки сообщества.

Для начала тестирования требуется создать проект тестирования, затем написать модули тестов, и после этого запустить проект тестирования [10].

Было создано 2 теста: «login.spec.js» для тестирования процесса авторизации и «chat.spec.js» для тестирования функционала чата и формы отзыва

о собеседнике. Процесс тестирования регистрации пользователей был произведён вручную, чтобы пронаблюдать, какие данные записываются в базу данных приложения в ходе регистрации.

Коды созданных модулей тестирования представлены в листинге 4 и листинге 5 соответственно.

Листинг 4 – код модуля «login.сpec.js»

```
const { chromium } = require('playwright');
const { expect } = require("playwright/test");

(async () => {
  const browser = await chromium.launch();
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('http://localhost:3000/login');

  // Убедитесь, что элементы существуют
  await page.waitForSelector('#login-input', { timeout: 10000 });
  await page.waitForSelector('#password-input', { timeout: 10000 });
  await page.waitForSelector('#login-button', { timeout: 10000 });

  // Test case: Login functionality
  await page.fill('#login-input', 'aboba');
  await page.fill('#password-input', '1');
  await page.click('#login-button');

  await expect(page).toHaveURL('http://localhost:3000/cabinet');

  console.log('Тест авторизации успешно пройден!');
  // Close the browser
  await browser.close();
})();
```

Листинг 5 – код модуля «chat.сpec.js»

```
const { chromium } = require('playwright');

(async () => {
  const browser = await chromium.launch();
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('http://localhost:3000/chat');

  // Функциональность кнопки смены собеседника
  await page.click('#getNextUserButton');
  await page.waitForSelector('#submit-form', { timeout: 10000 });

  // Функциональность формы отзыва
  await page.click('#submit-form');
  // Проверка на загрузку нового пользователя
  await page.waitForSelector('#new-user-profile', { timeout: 10000 });

  console.log('Тест чата успешно пройден!');
```

```
// Close the browser
await browser.close();
})();
```

Описание функционального тестирования разработанного приложения представлено в таблице 9.

Результаты автоматического и ручного функционального тестирования приложения приведены в приложении Б на рисунках Б.1-Б.2.

Таблица 9 – Описание функционального тестирования

Тестируемая ситуация	Описание теста	Тестовые данные	Ожидаемый результат	Наблюдаемый результат	Результат тестирования
1	2	3	4	5	6
Авторизация пользователя	Пользователь заходит на страницу «/login», вводит данные, совпадающие с данными из БД	Логин = «aboba» Пароль = «1»	Пользователь перенаправляется на страницу «/cabinet»	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (Рисунок Б.1)
Регистрация пользователя	Пользователь заходит на страницу «/register», вводит новые данные	Логин = «TestRegister» Пароль = «test» Имя = «Test»	Пользователь перенаправляется на страницу «/login», затем в БД приложения появляется новая запись в таблице users со значениями, введенными пользователем	Результат соответствует ожидаемому	Тест пройден (Рисунок Б.2)

Продолжение таблицы 9

1	2	3	4	5	6
Тестирование функционала страницы чата	Пользователь переходит на страницу «/chat», нажимает на кнопку «Следующий пользователь», заполняет форму отзыва о собеседнике и нажимает	Нет	При нажатии кнопки «Следующий пользователь», открывается форма отзыва о собеседнике, а после её отправки, форма становится не видна, а в чат подключается новый собеседник	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (Рисунок Б.1)

Осуществление функционального тестирования показало, что приложение функционирует корректно и не содержит явных ошибок и сбоев в работе. Данный результат является положительным результатом проведения тестирования.

8.3 Тестирование совместимости

Основная цель такого тестирования состоит в проверке того, что функциональность и внешний вид веб-приложения одинаково хороши в разных браузерах без значительных отклонений и ошибок [9]. Данное web-приложение было протестировано в самых популярных браузерах. Тестирование заключалось в открытии и сравнении всех основных страниц приложения: главной страница, страниц авторизации и регистрации, страницы личного кабинета, страницы чата. Результаты тестирования приложения приведены в приложении Б.

Проведённое тестирование показало, что во всех тестовых браузерах разработанное web-приложение функционирует корректно. Данный результат является положительным результатом проведения тестирования.

8.4 Тестирование пользовательского интерфейса

Для успешного проведения тестирования пользовательского интерфейса прежде всего следует выделить тестируемые элементы:

- окно (диалоговое окно, модальное окно, плавающее окно);
- меню (главное, всплывающее, контекстное, системное);
- виджеты/элементы управления/контролы (кнопка, радио-кнопка;
- чек-бокс, значок (иконка), список, панель инструментов, полоса прокрутки, ползунок);
- вкладка;
- элементы взаимодействия: курсор мыши, текстовый курсор, поинтер («ладошка»), курсор перетаскивания и др.

Также следует выделить основные задачи тестирования:

- проверить расположение, размер, цвет, ширину, длину элементов; возможность ввода букв или цифр;
- проверить размещение всех сообщений об ошибках, уведомлений (а также шрифт, цвет, размер, расположение и орфография текста);
- удостовериться, что использованный шрифт читабелен;
- выявить, переходит ли курсор из текстового в поинтер при наведении на активные элементы, выделяются ли выбранные элементы;
- проверить расположение и отображение всех элементов при различных разрешениях экрана, а также при изменении размера окна (проверить, появляется ли скролл);
- проверить текст на орфографические, пунктуационные ошибки;
- сделать унификацию дизайна (цвета, шрифты, текст сообщений, названия кнопок).

Результаты тестирования пользовательского интерфейса разработанного приложения представлен в таблице 10.

Таблица 10 – Результаты тестирования ПИ

Тестируемый параметр	Наблюдаемый результат	Результат теста
Орфография и пунктуация	Орфографических и пунктуационных ошибок не обнаружено	Тест пройден
Унификация дизайна	Все элементы единообразны на основных страницах Web-приложения	Тест пройден
Шрифт	На всех страницах используется одинаковый шрифт, размер шрифта единообразен для схожих элементов	Тест пройден
Адаптация элементов	Элементы успешно адаптируются при изменении размера экрана (рисунок Б.3)	Тест пройден
Элементы взаимодействия	Элементы взаимодействия функциональны и визуально изменяются при взаимодействии с ними (рисунок Б.4 и Б.5)	Тест пройден

Проведённое тестирование ПИ показало, что все элементы работают и визуально отображаются корректно. Данный результат является положительным результатом проведения тестирования.

8.5 Расчет метрик кода

Метрики кода - это количественные показатели, используемые для оценки качества, эффективности и сложности программного обеспечения. Анализ метрик кода помогает разработчикам и менеджерам проектов принимать

обоснованные решения, улучшать качество кода и повышать производительность команды.

Для данного Web-приложения в качестве метрик кода были взяты метрики SLOC (Source Lines of Code) и метрика Холстеда.

Source Lines of Code (SLOC) или количество строк кода, - это метрика оценки ПО, подсчитывающая физические и логические строки кода, а также комментарии. Проведем подсчет объема модуля «chatModel.js», используя данную метрику. Код модуля приведен в листинге 6.

Листинг 6 – код модуля «chatModel.js»

```
const sql = require('mssql');
const db = require('../config/dbConfig');

const getRandomUser = async () => {
  const pool = await db.connect();
  const result = await pool.request()
    .query('SELECT TOP 1 * FROM [users] WHERE status = \'chatting\'
ORDER BY NEWID() ');
  return result.recordset[0];
};

module.exports = {
  getRandomUser,
  // Other exports...
};
```

Для кода данного модуля показатели равны:

- физические строки кода (PLOC) - 12 строк;
- логические строки кода (LLOC) - 11 строк;
- комментарии – 1 строка;
- SLOC – 12 строк.

Теперь подсчитаем метрику Холстеда для данного модуля. Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- NUOprtr (Number of Unique Operators) - число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- NUOprnd (Number of Unique Operands) - число уникальных операндов программы (словарь операндов);

- Noprtr (Number of Operators) - общее число операторов в программе;
- Noprnd (Number of Operands) - общее число операндов в программе.

Операторы для данного модуля представлены в таблице 11.

Таблица 11 – Операторы модуля

Оператор	Количество вхождений
=	5
require	2
()	4
Async	1
Await	2
.	5
{ }	3
Query	1
Return	1

Операнды для данного модуля представлены в таблице 12.

Таблица 12 – Операнды модуля

Операнд	Количество вхождений
sql	1
db	2
getRandomUser	2
pool	2
db.connect	1
pool.request	1
Result	2

Продолжение таблицы 12

'SELECT TOP 1 * FROM [users] WHERE status = 'chatting' ORDER BY NEWID()'	1
Result.recordser[0]	1
module.exports	1

Словарь модуля рассчитывается по формуле:

$$n = n1 + n2 = 9 + 10 = 19 ,$$

где n – словарь модуля;

$n1$ – количество уникальных операторов;

$n2$ – количество уникальных операндов.

Длина модуля рассчитывается по формуле:

$$N = N1 + N2 = 22 + 14 = 36 ,$$

где N – длина модуля;

$N1$ – общее количество операторов;

$N2$ – общее количество операндов.

Объём процедуры рассчитывается по формуле:

$$V = N * \log_2(n) =$$

Сложность процедуры рассчитывается по формуле:

$$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right) = \left(\frac{9}{2}\right) * \left(\frac{14}{10}\right) = 6,3$$

Модуль имеет компактный и хорошо структурированный код, что способствует его легкости понимания и поддержки. Однако, метрика Холстеда показывает, что для его реализации и тестирования потребуются значительные усилия, что связано с использованием асинхронных операций и взаимодействием с базой данных. Это важно учитывать при планировании задач по разработке и тестированию данного модуля.

Заключение

В ходе выполнения выпускной квалификационной работы был проведен анализ предметной области, в рамках которого был рассмотрен процесс создания Web-платформы по изучению иностранных языков. В процессе анализа были исследованы существующие решения реализации поставленной задачи и выявлены основные функции и структура разрабатываемого приложения.

Кроме того, был проведен детальный анализ системы с использованием различных диаграмм UML: диаграммы вариантов использования, диаграммы состояний, диаграммы классов, диаграммы последовательности, диаграммы кооперации и диаграммы компонентов. Эти диаграммы помогли наглядно представить архитектуру приложения, взаимодействие компонентов и процессы, происходящие в нём.

Для реализации Web-платформы были выбраны современные технологии и средства разработки, такие как React для пользовательского интерфейса и Node.js с использованием Express для серверной части.

В результате была разработана Web-платформа, позволяющая пользователям находить собеседников и вести с ними чат на выбранных языках. В системе также реализована система рейтингов и отзывов, что помогает улучшить взаимодействие пользователей и повысить качество общения.

Тестирование платформы состояло из выбора метода тестирования и выполнения тестов. Результаты тестирования показали корректность работы основных функций приложения. Web-платформа была проверена на совместимость с различными браузерами, что гарантирует её стабильную работу на большинстве популярных платформ. Также было проведено тестирование пользовательского интерфейса, которое показало корректность и удобство визуальной составляющей разработанной Web-платформы.

В дальнейшем разработанная платформа может быть усовершенствована добавлением новых функций, таких как улучшенная система рекомендаций собеседников, и дополнительных возможностей для пользователей, что сделает платформу еще более полезной и удобной в использовании.

Список использованных источников

1. Проектирование информационных систем [Электронный ресурс]. URL: <https://moodle.kstu.ru/course/view.php?id=4638&> (Дата обращения 29.05.2024)
2. Что такое интерфейс пользователя? [Электронный ресурс]. URL: <https://www.internet-technologies.ru/articles/newbie/polzovatelskiy-interfeys.html> (Дата обращения: 30.05.2024).
3. Никитина Е.Б. Функционально-стоимостный анализ - П., 2021. - 100 с.
4. End-to-end тестирование React приложения с помощью Playwright [Электронный ресурс]. URL: <https://habr.com/ru/articles/791618/> (Дата обращения 01.06.2024)
5. Простое руководство к диаграммам развертывания UML [Электронный ресурс]. URL: <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/> (Дата обращения: 30.05.2024).
6. Простое руководство по диаграммам компонентов [Электронный ресурс]. URL: <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/> (Дата обращения: 30.05.2024).
7. Четыре типа диаграмм в помощь разработчику или зачем нужен UML [Электронный ресурс]. URL: <https://worksolutions.ru/blog/why-use-uml/> (Дата обращения 01.06.2024)
8. Как настроить Node.js Express сервер для React [Электронный ресурс]. URL: <https://habr.com/ru/articles/746906/> (Дата обращения: 15.05.2024).
9. Почему необходимо выполнять тестирование пользовательского интерфейса? [Электронный ресурс]. URL: <https://testmatick.com/ru/our-qa-services/user-interface-testing-services/#> (Дата обращения: 30.05.2024).
10. Начало работы с Playwright [Электронный ресурс]. URL: <https://habr.com/ru/articles/597293/> (Дата обращения: 30.05.2024).

11. React.js: понятное руководство для начинающих [Электронный ресурс]. URL: <https://habr.com/ru/companies/ruvds/articles/428077/> (Дата обращения: 30.05.2024).

12. Автоматическое тестирование в браузере [Электронный ресурс]. URL: <https://habr.com/ru/companies/ozontech/articles/592295/> (Дата обращения: 29.05.2024).

Приложение А

Листинг программы

Листинг А.1 – код модуля «dbConfig.js»

```
const sql = require('mssql');

const dbConfig = {
  user: 'Diplom',
  password: '1337',
  server: 'DESKTOP-000BF2B',
  database: 'languageNew',
  options: {
    encrypt: false, // for azure
    trustServerCertificate: true // change to false for production
  }
};

let pool;

const connect = async () => {
  if (!pool) {
    pool = await sql.connect(dbConfig);
  }
  return pool;
};

module.exports = {
  connect
};
```

Листинг А.2 – код модуля «authController.js»

```
const userModel = require('../models/userModel');

const login = async (req, res) => {
  const { login, password } = req.body;
  const user = await userModel.getUserByLogin(login);
  if (user && user.password === password) {
    res.json({ success: true, userId: user.id_user });
  } else {
    res.json({ success: false, message: 'Invalid login or password',
    login: login, password: password, user: user});
  }
};

const register = async (req, res) => {
  const { name, login, password } = req.body;
  const result = await userModel.createUser({ name, login, password });
  if (result) {
    res.json({ success: true });
  } else {
    res.json({ success: false, message: 'Registration failed' });
  }
};

module.exports = {
  login,
  register
};
```

Листинг А.3 – код модуля «chatController.js»

```

const chatModel = require('../models/chatModel');
const {getUserById} = require("../models/userModel");

const getChatUsers = async (req, res) => {
  try {
    const user1 = await getUserById(req.params.id);
    if (!user1) {
      return res.status(404).json({ message: "User1 not found" });
    }

    let user2 = null;
    do {
      user2 = await chatModel.getRandomUser();
    } while (user2 && user2.id_user === user1.id_user);

    res.json({ user1, user2 });
  } catch (error) {
    console.error('Error fetching chat users:', error);
    res.status(500).json({ message: 'Server error: Error fetching chat users' });
  }
};

const getNextUser = async (req, res) => {
  try {
    const currentUserId = parseInt(req.params.currentUserId, 10);
    let user = null;
    do {
      user = await chatModel.getRandomUser();
    } while (user && user.id_user === currentUserId);

    if (!user) {
      return res.status(404).json({ message: 'No user found' });
    }
    res.json(user);
  } catch (error) {
    console.error('Error fetching random user:', error);
    res.status(500).json({ message: 'Server error: Error fetching random user' });
  }
};

module.exports = {
  getChatUsers,
  getNextUser
};

```

Листинг А.4 – код модуля «messageController.js»

```

const messageModel = require('../models/messageModel');

const createMessage = async (req, res) => {
  try {
    const message = await messageModel.createMessage(req.body);
    res.status(201).json(message);
  }
};

```

```

    } catch (error) {
      console.error('Error creating message:', error);
      res.status(500).json({ message: 'Server error' });
    }
  };

const getMessages = async (req, res) => {
  try {
    const { user1Id, user2Id } = req.params;
    const messages = await messageModel.getMessages(user1Id,
user2Id);
    res.json(messages);
  } catch (error) {
    console.error('Error fetching messages:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

module.exports = {
  createMessage,
  getMessages,
};

```

Листинг А.5 – код модуля «dbConfig.js»

```

const userModel = require('../models/userModel');
const {addReview} = require("../models/userModel");

const getUser = async (req, res) => {
  const user = await userModel.getUserById(req.params.id);
  res.json(user);
};

async function updateUser(req, res) {
  const { id_user, name } = req.body;
  try {
    const result = await userModel.updateUser(id_user, { name });
    res.status(200).json({ success: true, message: 'User updated
successfully', data: result });
  } catch (error) {
    res.status(500).json({ success: false, message: 'Update failed',
error: error.message });
  }
}

const updateUserStatus = async (req, res) => {
  const status = req.body.status;
  const userId = req.body.id;

  try {
    const result = await userModel.updateUserStatusInDB(userId,
status);
    if (result) {
      res.status(200).json({ message: 'Status updated successfully
'+userId+status });
    } else {
      res.status(404).json({ message: 'User not
found'+req.params.id+req.params.status+req.body });
    }
  } catch (error) {

```

```

        console.error('Error updating user status:', error);
        res.status(500).json({ message: 'Server error: Error updating
user status' });
    }
};

const rateUser = async (req, res) => {
    const userId = req.body.idUser;
    const ratedUserId = req.body.ratedUserId;
    try {
        const result = await userModel.addReview(userId, ratedUserId,
req.body.rating, req.body.comment);
        if (result) {
            res.status(200).json({ message: 'Rating added successfully'+"
"+ratedUserId+" rat "+req.body.rating });
        } else {
            res.status(500).json({ message: 'Failed to add rating'+"
"+ratedUserId+" rat "+req.body.rating });
        }
    } catch (error) {
        console.error('Error adding rating:', error);
        res.status(500).json({ message: 'Server error: Error adding
rating', error });
    }
};

module.exports = {
    getUser,
    updateUser,
    updateUserStatus,
    rateUser
};

```

Листинг А.6 – код модуля «chatModel.js»

```

const sql = require('mssql');
const db = require('../config/dbConfig');

const getRandomUser = async () => {
    const pool = await db.connect();
    const result = await pool.request()
        .query('SELECT TOP 1 * FROM [users] WHERE status = \'chatting\'
ORDER BY NEWID() ');
    return result.recordset[0];
};

module.exports = {
    getRandomUser,
    // Other exports...
};

```

Листинг А.7 – код модуля «messageModel.js»

```

const sql = require('mssql');
const dbConfig = require('../config/dbConfig');

const createMessage = async (message) => {
    const pool = await sql.connect(dbConfig);

```

```

    const result = await pool.request()
      .input('senderId', sql.Int, message.senderId)
      .input('receiverId', sql.Int, message.receiverId)
      .input('content', sql.NVarChar, message.content)
      .query(`
        INSERT INTO messages (senderId, receiverId, content)
        VALUES (@senderId, @receiverId, @content)
      `);
    return result;
  };

const getMessages = async (user1Id, user2Id) => {
  const pool = await sql.connect(dbConfig);
  const result = await pool.request()
    .input('user1Id', sql.Int, user1Id)
    .input('user2Id', sql.Int, user2Id)
    .query(`
      SELECT * FROM messages
      WHERE (senderId = @user1Id AND receiverId = @user2Id)
        OR (senderId = @user2Id AND receiverId = @user1Id)
      ORDER BY id ASC
    `);
  return result.recordset;
};

module.exports = {
  createMessage,
  getMessages,
};

```

Листинг А.8 – код модуля «userModel.js»

```

const sql = require('mssql');
const dbConfig = require('../config/dbConfig');

const getUserById = async (id) => {
  const pool = await sql.connect(dbConfig);
  let result = await pool.request()
    .input('id', sql.Int, id)
    .query('SELECT * FROM [users] WHERE id_user = @id');
  return result.recordset[0];
};

const getUserByLogin = async (login) => {
  const pool = await sql.connect(dbConfig);
  let result = await pool.request()
    .input('login', sql.NVarChar, login)
    .query('SELECT * FROM [users] WHERE login = @login');
  return result.recordset[0];
};

async function createUser({ name, login, password }) {
  try {
    let pool = await sql.connect(dbConfig);
    let result = await pool.request()
      .input('name', sql.NVarChar, name)
      .input('login', sql.NVarChar, login)
      .input('password', sql.NVarChar, password)

```

```

        .query('INSERT INTO [users] (name, login, password) VALUES
(@name, @login, @password)');
        return result;
    } catch (error) {
        throw error;
    }
}

const updateUserStatusInDB = async (userId, status) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request()
            .input('userId', sql.Int, userId)
            .input('status', sql.VarChar, status)
            .query('UPDATE users SET status = @status WHERE id_user =
@userId');

        return result.rowsAffected[0] > 0;
    } catch (error) {
        console.error('Error updating user status in DB:', error);
        throw error;
    }
};

// Функция для обновления пользователя
async function updateUser(id_user, { name }) {
    try {
        let pool = await sql.connect(dbConfig);
        let result = await pool.request()
            .input('id_user', sql.Int, id_user)
            .input('name', sql.NVarChar, name)
            .query('UPDATE users SET name = @name WHERE id_user =
@id_user');
        return result;
    } catch (error) {
        throw error;
    }
}

const addReview = async (userId, ratedUserId, rating, comment) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request()
            .input('userId', sql.Int, userId)
            .input('ratedUserId', sql.Int, ratedUserId)
            .input('rating', sql.Int, rating)
            .input('comment', sql.NVarChar, comment)
            .query('INSERT INTO reviews (userId, ratedUserId, rating,
comment) VALUES (@userId, @ratedUserId, @rating, @comment)');
        if (result) {
            await updateAverageRating(ratedUserId); // После добавления
оценки обновляем средний рейтинг
            return true;
        }
        return false;
    } catch (error) {
        console.error('Error adding rating:', error);
        throw error;
    }
}

```

```

    }
  };

  const updateAverageRating = async (ratedUserId) => {
    try {
      const pool = await sql.connect(dbConfig);
      const result = await pool.request()
        .input('ratedUserId', sql.Int, ratedUserId)
        .query('SELECT AVG(rating) AS averageRating FROM reviews
WHERE ratedUserId = @ratedUserId');

      const averageRating = result.recordset[0].averageRating || 0;

      await pool.request()
        .input('ratedUserId', sql.Int, ratedUserId)
        .input('averageRating', sql.Float, averageRating)
        .query('UPDATE users SET rating = @averageRating WHERE
id_user = @ratedUserId');
    } catch (error) {
      console.error('Error updating average rating:', error);
      throw error;
    }
  };
  // Other user-related methods...

  module.exports = {
    getUserById,
    createUser,
    getUserByLogin,
    updateUser,
    updateUserStatusInDB,
    updateAverageRating,
    addReview
  };
};

```

Листинг А.9 – код модуля «authRoutes.js»

```

const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/login', authController.login);
router.post('/register', authController.register);

module.exports = router;

```

Листинг A.10 – код модуля «chatRoutes.js»

```
const express = require('express');
const router = express.Router();
const chatController = require('../controllers/chatController');

router.get('/:id', chatController.getChatUsers);
router.get('/random-user/:currentUserId', chatController.getNextUser);

module.exports = router;
```

Листинг A.11 – код модуля «messageRoutes.js»

```
const express = require('express');
const router = express.Router();
const messageController = require('../controllers/messageController');

router.post('/', messageController.createMessage);
router.get('/:user1Id/:user2Id', messageController.getMessages);

module.exports = router;
```

Листинг A.12 – код модуля «userRoutes.js»

```
const express = require('express');
const router = express.Router();
const messageController = require('../controllers/messageController');

router.post('/', messageController.createMessage);
router.get('/:user1Id/:user2Id', messageController.getMessages);

module.exports = router;
```

Листинг A.13 – код модуля «app.js»

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const authRoutes = require('./routes/authRoutes');
const userRoutes = require('./routes/userRoutes');
const chatRoutes = require('./routes/chatRoutes');
const app = express();
const messageRoutes = require('./routes/messageRoutes');

app.use(cors());
app.use(bodyParser.json());
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/chat', chatRoutes);
app.use('/api/messages', messageRoutes);

module.exports = app;
```

Листинг A.14 – код модуля «server.js»

```
const app = require('./app');
const db = require('./config/dbConfig');
```

```
const PORT = process.env.PORT || 5000;

db.connect().then(() => {
  app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
  });
}).catch(err => {
  console.error('Failed to connect to the database:', err);
});
```

Листинг A.15 – код модуля «index.html»

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>ПриветМир!</title>
</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>
<div id="root"></div>
</body>
</html>
```

Листинг A.16 – код модуля «api.js»

```
import axios from 'axios';

const API_URL = 'http://localhost:5000/api';

export const login = (login, password) => {
  return axios.post(`${API_URL}/auth/login`, { login, password });
};

export const register = (name, login, password) => {
  return axios.post(`${API_URL}/auth/register`, { name, login, password });
};

export const getUser = (id) => {
  return axios.get(`${API_URL}/users/${id}`);
};

export const updateUser = (id, data) => {
  return axios.put(`${API_URL}/users/${id}`, data);
};

export const getChatUsers = (id) => {
  return axios.get(`${API_URL}/chat/${id}`);
};

export const getNextUser = (id, id2) => {
  return axios.get(`${API_URL}/chat/random-user/${id}`);
};
```

```

export const sendMessage = (message) => {
  return axios.post(`${API_URL}/messages`, message);
};

export const getMessages = (user1Id, user2Id) => {
  return axios.get(`${API_URL}/messages/${user1Id}/${user2Id}`);
};

export const updateUserStatus = (id, status) => {
  return axios.put(`${API_URL}/users/status/${id}`, { id, status });
};

export const rateUser = (idUser, ratedUserId, rating, comment) => {
  return axios.put(`${API_URL}/users/rating/${idUser}`, { idUser,
    ratedUserId, rating, comment });
};

```

Листинг А.17 – код модуля «CabinetPage.js»

```

import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { getUser, updateUser } from '../api/api';
import { Button, Flex, Input, Checkbox, Divider } from 'antd';

const CheckboxGroup = Checkbox.Group;
const plainOptions = ['Английский', 'Русский', 'Французский'];

const CabinetPage = () => {
  const [user, setUser] = useState(null);
  const [name, setName] = useState('');
  const [rating, setRating] = useState('');
  const navigate = useNavigate('');

  useEffect(() => {
    const userId = localStorage.getItem('userId');
    if (userId) {
      getUser(userId).then(response => {
        setUser(response.data);
        setName(response.data.name);
        setRating(response.data.rating);
      });
    }
  }, []);

  const handleUpdate = () => {
    const userId = localStorage.getItem('userId');
    updateUser(userId, { name }).then(() => {
      // Handle successful update
    });
  };

  const handleChat = () => {
    navigate('/chat');
  };

  return (
    <div style={{

```

```

        backgroundImage:
`url(${require('../assets/img/background.jpg')})`,
        backgroundSize: '100% 100%',
        color: "white",
        height: '90vh',
        width: '100%'
    }}>
    <h1>Личный Кабинет</h1>
    <Flex align='center' justify='start' vertical gap='middle'
        style={{width: '100%', stroke: 'black', strokeWidth:
'thin', backgroundColor: 'rgb(0 0 0 / 30%)'}}>
        <h1>Имя</h1>
        <Input style={{width: 'auto'}} size='middle' type="text"
value={name}
            onChange={ (e) => setName(e.target.value)} />
        <Button onClick={handleUpdate}>Изменить имя</Button>
        {rating === null ? (
            <h1>Ваш рейтинг: Отсутствует</h1>
        ) : (
            <h1>Ваш рейтинг: {rating}</h1>
        )}
        <h1>Языки общения</h1>
        <CheckboxGroup options={plainOptions}
style={{backgroundColor: 'white'}} />
        <Button onClick={handleChat}>Начать чат</Button>
    </Flex>
</div>
);
};

export default CabinetPage;

```

Листинг А.18 – код модуля «ChatPage.js»

```

import React, { useState, useEffect } from 'react';
import { getChatUsers, getNextUser, updateUserStatus } from '../api/api';
import { getMessages, sendMessage } from '../api/api';
import RatingForm from './RatingForm';
import { Button, Flex, Space, Input } from 'antd';
import axios from 'axios';

const videosContext = require.context('../assets/video', false,
/\.mp4$/);
const videos = videosContext.keys().map(videosContext);
const getRandomVideo = () => {
    const randomIndex = Math.floor(Math.random() * videos.length);
    return videos[randomIndex];
};

const ChatPage = () => {
    const [user1, setUser1] = useState({});
    const [user2, setUser2] = useState({});
    const [messages, setMessages] = useState([]);
    const [newMessage, setNewMessage] = useState('');
    const userId = localStorage.getItem('userId');
    const [showRatingForm, setShowRatingForm] = useState(false);
    const [ratedUser, setRatedUser] = useState({});
    const [videoSrc, setVideoSrc] = useState(getRandomVideo())

```

```

    useEffect(() => {
      const userId = localStorage.getItem('userId');
      if (userId) {
        // Обновление статуса на "chatting" при монтировании
        КОМПОНЕНТА
        updateUserStatus(userId, 'chatting');
        fetchChatUsers();
      }
      // Обновление статуса на "notChatting" при размонтировании
      КОМПОНЕНТА
      return () => {
        updateUserStatus(userId, 'notChatting');
      };
    }, [userId]);

    const fetchChatUsers = async () => {
      const response = await getChatUsers(userId);
      setUser1(response.data.user1);
      if (response.data.user2) {
        setUser2(response.data.user2);
        fetchMessages(response.data.user1.id_user,
        response.data.user2.id_user);
      } else {
        setTimeout(fetchChatUsers, 2000); // Попробовать снова через
        2 секунды
      }
    };

    useEffect(() => {
      if (user1 && user2) {
        const fetchMessages = async () => {
          try {
            const response = await getMessages(user1.id_user,
            user2.id_user);
            setMessages(response.data);
          } catch (error) {
            console.error('Error fetching messages:', error);
          }
        };

        fetchMessages(); // Fetch messages when component mounts

        const intervalID = setInterval(fetchMessages, 2000); // Fetch
        messages every 2 seconds

        return () => clearInterval(intervalID); // Clear interval on
        component unmount
      }
    }, [user1, user2]);

    const fetchMessages = async (user1Id, user2Id) => {
      try {
        const response = await getMessages(user1Id, user2Id);
        setMessages(response.data);
        console.log(messages)
      }
    }
  }
}

```

```

    } catch (error) {
      console.error('Failed to fetch messages:', error);
    }
  };

  function handleNextButton() {
    setShowRatingForm(true);
    console.log("chat page id`s = ", user1.id_user, " ",
user2.id_user);
    setRatedUser(user2.id_user);
    getNextUserButton()
  }

  const getNextUserButton = async () => {
    try {
      const userId = user2.id_user;
      const mainUserId = user1.id_user;
      getNextUser(userId, mainUserId).then(response => {
        if (response.data.id_user !== mainUserId &&
response.data.id_user !== userId){
          setUser2(response.data);
          fetchMessages(user1.id_user, user2.id_user);
          setVideoSrc(getRandomVideo())
        }
        else {
          getNextUserButton();
        }
      });
    } catch (error) {
      console.error('Failed to fetch next user:', error);
    }
  };

  const handleSendMessage = async () => {
    if (user1 && user2) try {
      const message = {
        senderId: user1.id_user,
        receiverId: user2.id_user,
        content: newMessage,
      };
      await sendMessage(message);
      setNewMessage('');
      fetchMessages(user1.id_user, user2.id_user); // обновляем
сообщения после отправки
    } catch (error) {
      console.error('Failed to send message:', error);
    }
  };

  if (!user1 || user2===null || !user2) {
    return <div>Loading...</div>;
  }
  else{
    return (
      <div style={{

```

```

        backgroundImage:
`url(${require('../assets/img/background.jpg')})`,
        backgroundSize: '100% 100%',
        color: "white",
        height: '90vh',
        width: '100%'
    }}>
    <h1>Chat</h1>
    <p>Your name: {user1.name}</p>
    <div/>
    {showRatingForm ? (
        <RatingForm userId={user1.id_user}
ratedUserId={ratedUser} setShowRatingForm={setShowRatingForm}/>
    ) : (
        <div>
            <p id='new-user-profile'>Chat with: {user2.name} (Rating:
{user2.rating})</p>
            <Flex gap='small' align='center' justify='center'>
                <div style={{display: 'grid', justifyContent: 'center',
alignItems: 'center'}}>
                    <video width="420" height="300" autoPlay muted loop>
                        <source src={videoSrc} type="video/mp4"/>
                    </video>
                    <div>
                        <div style={{background: 'white', backgroundSize:
'100% 100%', color: "black", width: 'auto'}}>
                            {messages.map((msg) => (
                                <div key={msg.id}>
                                    <strong>{msg.senderId === user1.id_user ?
'You' : user2.name}:</strong> {msg.content}
                                </div>
                            ))}
                        </div>
                        <Space gap='middle'>
                            <Input
                                type="text"
                                value={newMessage}
                                onChange={(e) => setNewMessage(e.target.value)}
                            />
                            <Button type = 'primary'
onClick={handleSendMessage}>Отправить сообщение</Button>
                            <Button id='getNextUserButton'
onClick={handleNextButton}>Следующий пользователь</Button>
                        </Space>
                    </div>
                </div>
            </Flex>
        </div>
    )}

    {/* Chat UI goes here */}
</div>
);
}
};

export default ChatPage;

```

Листинг А.19 – код модуля «LoginPage.js»

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { login } from '../api/api';
import { Button, Space, Input } from 'antd';

const LoginPage = () => {
  const [loginInput, setLoginInput] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleLogin = () => {
    login(loginInput, password).then(response => {
      if (response.data.success) {
        localStorage.setItem('userId', response.data.userId);
        navigate('/cabinet');
      } else {
        console.log(response.message, response.login,
response.password)
        console.log(response.user);
      }
    });
  };

  return (
    <div style = {{backgroundImage:
`url(${require('../assets/img/background.jpg') })`, backgroundSize: '100%
100%', color:"white", height: '90vh',
width: '100%'}}>
      <h1>Login</h1>
      <Space align="center">
        Логин
        <Input id="login-input" type="text" value={loginInput}
onChange={e => setLoginInput(e.target.value)} />
        Пароль
        <Input id="password-input" type="password" value={password}
onChange={e => setPassword(e.target.value)} />
        <Button id="login-button"
onClick={handleLogin}>Login</Button>
      </Space>
    </div>
  );
};

export default LoginPage;
```

Листинг А.20 – код модуля «MainPage.js»

```
import React from 'react';
import { Link } from 'react-router-dom';
import { Button, Space, Flex } from 'antd';

const MainPage = () => {
  return (
    <div style = {{backgroundImage:
`url(${require('../assets/img/background.jpg') })`, backgroundSize: '100%
100%', height: '90vh',
```

```

width: '100%'}}>
<h1 style={{color: "white"}}>ПриветМир!</h1>
<Flex gap='middle' align='center' justify='center' vertical>
<text style={{
  color: "white", fontSize: 30, backgroundColor: 'rgb(0 0 0
/ 30%)'
}}><b> ПриветМир! </b>
  - это платформа, которая позволяет людям по всему миру
общаться между собой и изучать иностранные языки.
  Присоединяйся!
  <p/>
  <Space>
  <Button type='primary'>
    <Link to="/login" style={{fontSize: 20, textAlign:
"center"}}>>Login
    </Link>
  </Button>
  <Button>
    <Link to="/register"
      style={{fontSize: 20, textAlign: "center"}}>>
      Register
    </Link>
  </Button>
  </Space>
</text>
</Flex>
</div>
);
};

export default MainPage;

```

Листинг A.21 – код модуля «RatingForm.js»

```

import React, { useState } from 'react';
import axios from 'axios';
import { rateUser } from "../api/api";
import { Space, Flex } from 'antd';

const RatingForm = ({ userId, ratedUserId, setShowRatingForm }) => {
  const [rating, setRating] = useState(1);
  const [comment, setComment] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await rateUser(userId, ratedUserId, rating,
comment);
      console.log('Review submitted successfully', response.data);
      // Дополнительные действия после успешного сохранения отзыва,
например, обновление интерфейса
    } catch (error) {
      console.error('Failed to submit review', error);
    }
    console.log('form submitted'+ " rated user = "+ratedUserId);
    setShowRatingForm(false);
  };
};

```

```

    return (
      <Flex gap='small' align='center' justify='center'
vertical='true' style={{stroke:'black', strokeWidth:'thin',
backgroundColor: 'rgb(0 0 0 / 30%)', width:'auto' }}>
        <h2>Leave a Review</h2>
        <form onSubmit={handleSubmit}>
          <Space gap='middle' direction='vertical' align='start'>
            <div style={{display: 'ruby', justifyContent:
'space-between'}}>
              <h2>Rating: </h2>
              <select value={rating} onChange={ (e) =>
setRating(e.target.value)}>
                <option value="1">1</option>
                <option value="2">2</option>
                <option value="3">3</option>
                <option value="4">4</option>
                <option value="5">5</option>
              </select>
            </div>
            <br />
            <label>
              <h2>Comment (optional):</h2>
              <textarea value={comment} onChange={ (e) =>
setComment(e.target.value)} />
            </label>
            <br />
            <button id='submit-form' type="submit">Отправить</button>
          </Space>
        </form>
      </Flex>
    );
  };

export default RatingForm;

```

Листинг А.22 – код модуля «RegisterPage.js»

```

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { register } from '../api/api';
import { Button, Space, Input } from 'antd';

const RegisterPage = () => {
  const [name, setName] = useState('');
  const [loginInput, setLoginInput] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleRegister = () => {
    register(name, loginInput, password).then(response => {
      if (response.data.success) {
        navigate('/login');
      } else {
        console.log(response.data);
      }
    });
  };
};

```

```

    return (
      <div style = {{backgroundImage:
`url(${require('../assets/img/background.jpg') })`, backgroundSize: '100%
100%', color:"white", height: '90vh',
      width: '100%'}}>
        <h1>Register</h1>
        <Space align="center">
          Имя
          <Input type="text" value={name} onChange={ (e) =>
setName(e.target.value)} />
          Логин
          <Input type="text" value={loginInput} onChange={ (e) =>
setLoginInput(e.target.value)} />
          Пароль
          <Input type="password" value={password} onChange={ (e) =>
setPassword(e.target.value)} />
          <Button onClick={handleRegister}>Register</Button>
        </Space>
      </div>
    );
  };

export default RegisterPage;

```

Листинг А.23 – код модуля «App.js»

```

import React, { useState } from 'react';
import { Route, Routes } from 'react-router-dom';
import MainPage from '../components/MainPage';
import CabinetPage from '../components/CabinetPage';
import LoginPage from '../components/LoginPage';
import RegisterPage from '../components/RegisterPage';
import ChatPage from '../components/ChatPage';

function App() {
  const [user, setUser] = useState(null);
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<MainPage />} />
        <Route path="/cabinet" element={<CabinetPage />} />
        <Route path="/login" element={<LoginPage />} />
        <Route path="/register" element={<RegisterPage />} />
        <Route path="/chat" element={<ChatPage user={user}/>} />
      </Routes>
    </div>
  );
}

export default App;

```

Листинг А.24 – код модуля «index.js»

```

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

```

```
ReactDOM.render(  
  <BrowserRouter>  
    <App />  
  </BrowserRouter>,  
  document.getElementById('root')  
);
```

Приложение Б

Результаты тестирования

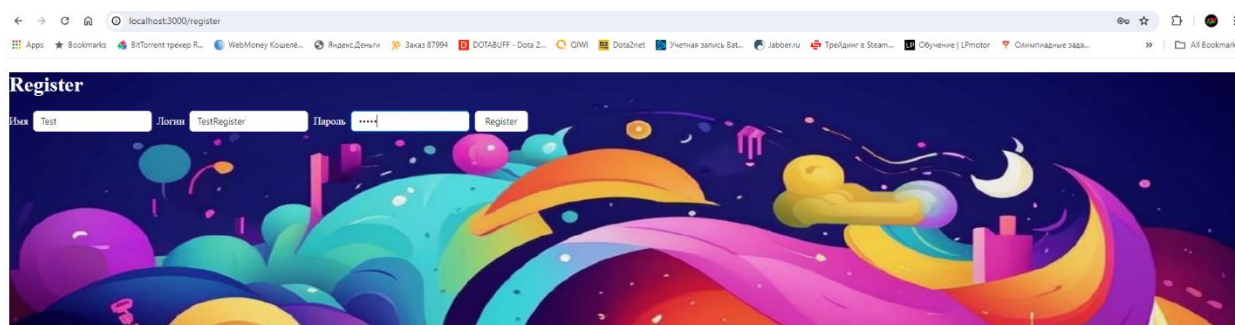
```

PS C:\Users\Egor\Desktop\diplom\project\playwright> npm run test
> playwright@1.0.0 test
> node tests/example/login.spec.js && node tests/example/chat.spec.js

Тест авторизации успешно пройден!
Тест чата успешно пройден!
PS C:\Users\Egor\Desktop\diplom\project\playwright>

```

Рисунок Б.1 – Результаты выполнения автоматических тестов



id_user	name	login	password	photo	status	known_languages	rating
1	Aboba	Aboba	1	NULL	chatting	NULL	4
2	Goben	gob	1	NULL	chatting	NULL	3
3	Rand1	random1	1	NULL	chatting	NULL	4
4	Random2	random2	1	NULL	notChatting	NULL	4
5	test	test	test	NULL	chatting	NULL	0
6				NULL	NULL	NULL	0
7	1	1	1	NULL	chatting	NULL	0
8	Test	TestRegister	teest	NULL	NULL	NULL	0

Рисунок Б.2 – Тестирование процесса регистрации

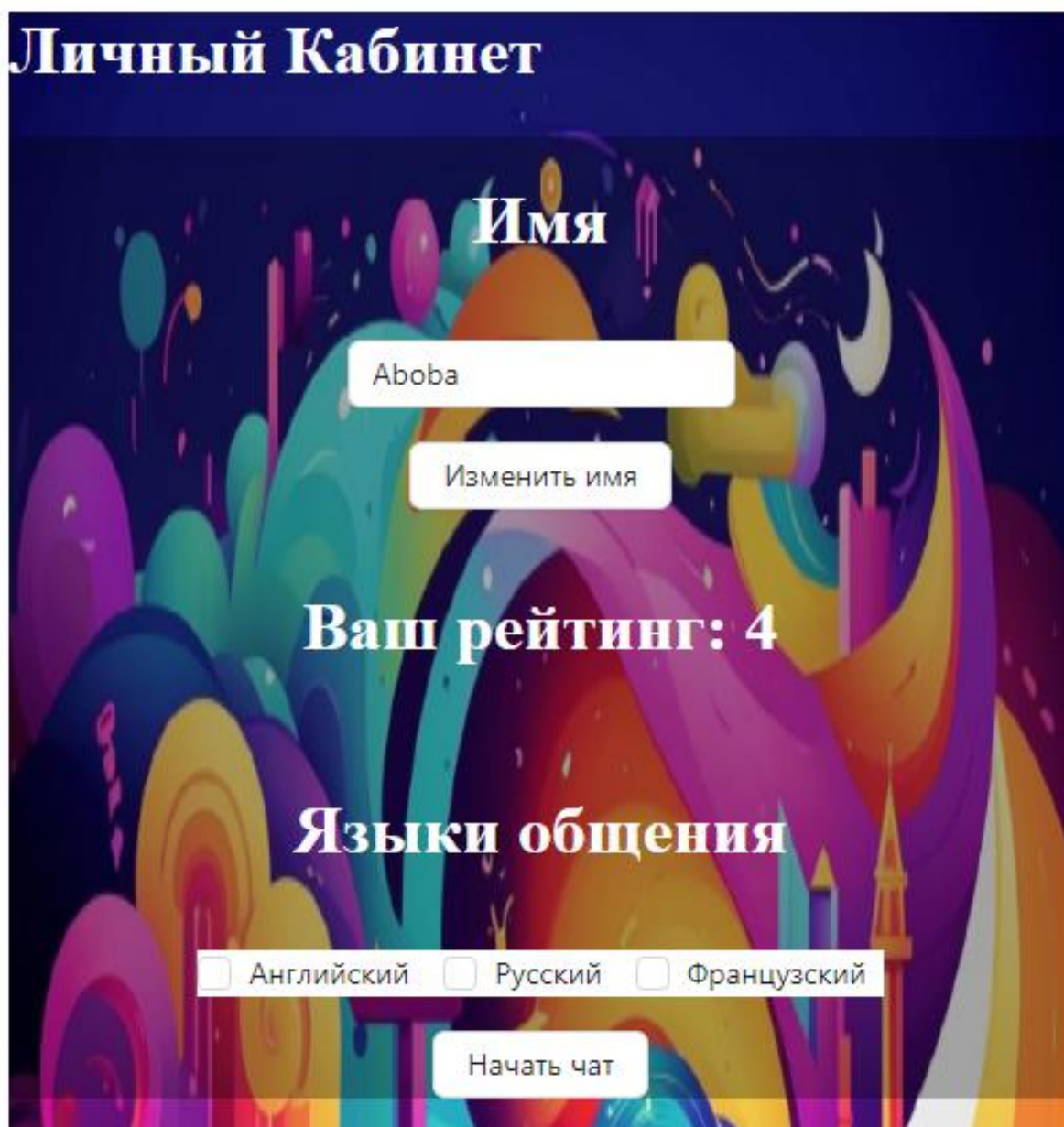


Рисунок Б.3 – Тестирование адаптации элементов к размеру экрана

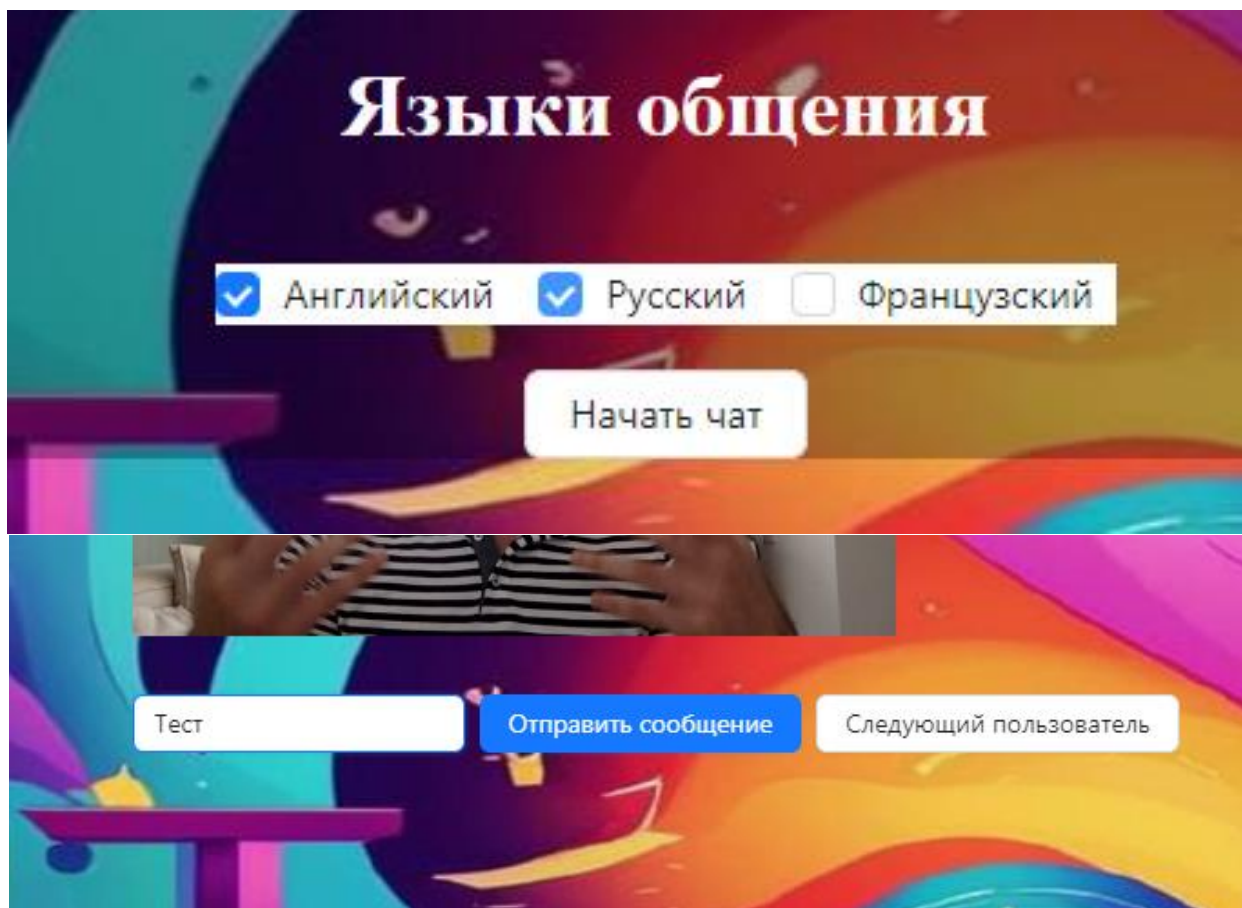


Рисунок Б.4 – Выбор в поле CheckBox и заполнение поля Input

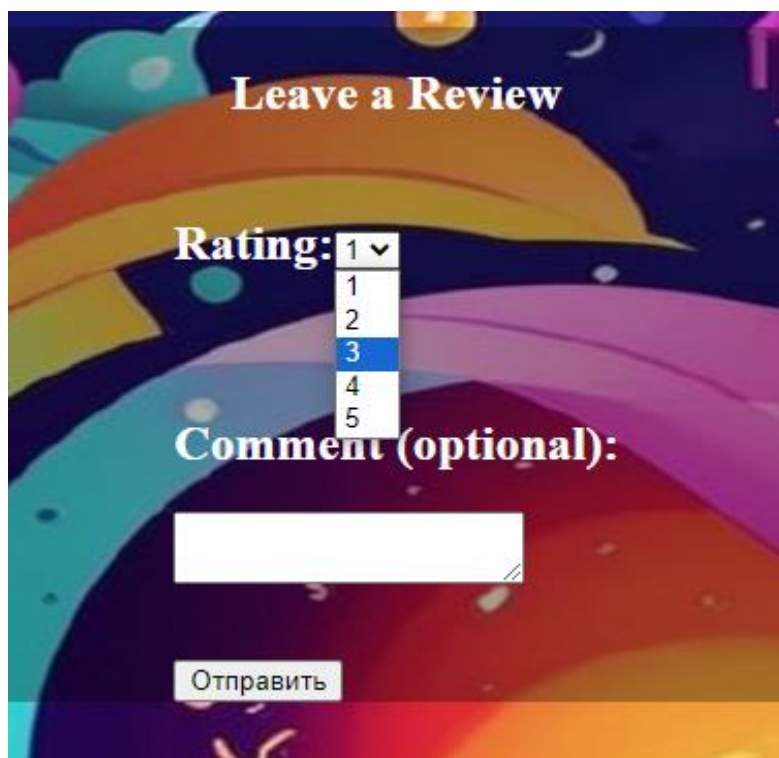


Рисунок Б.5 – Выбор значения из выпадающего списка

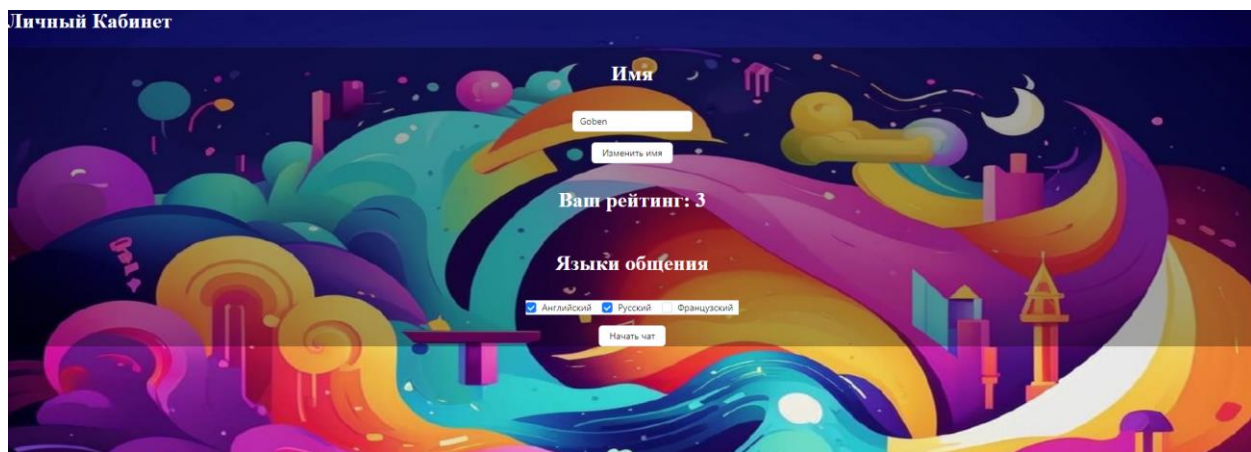


Рисунок Б.6 – Открытие личного кабинета в браузере «GoogleChrome»

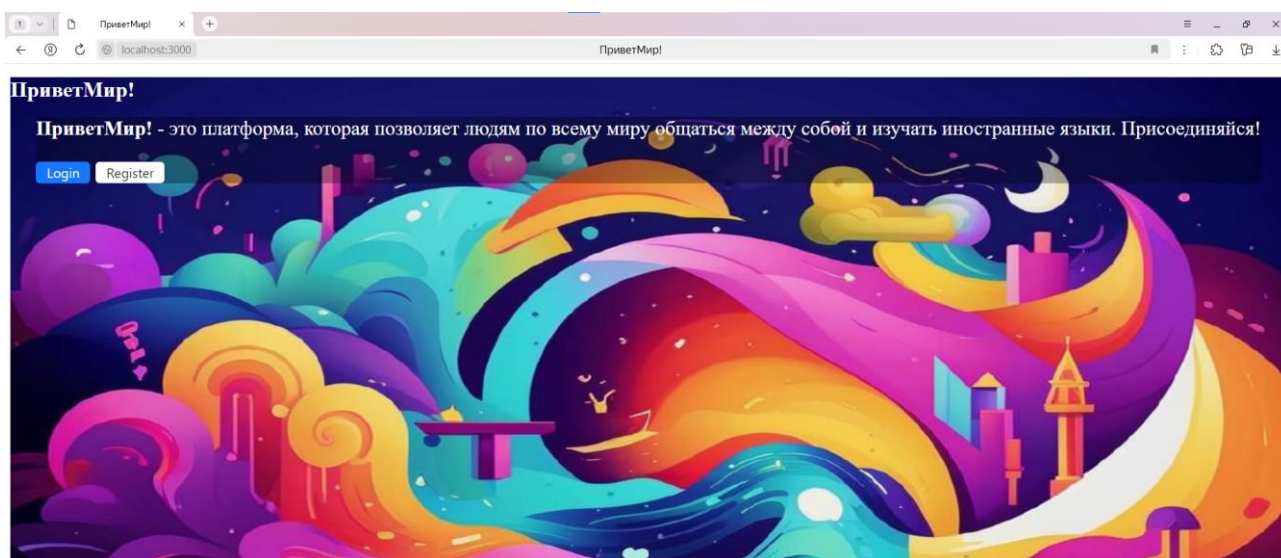


Рисунок Б.7 – Открытие главной страницы в браузере «Яндекс Браузер»

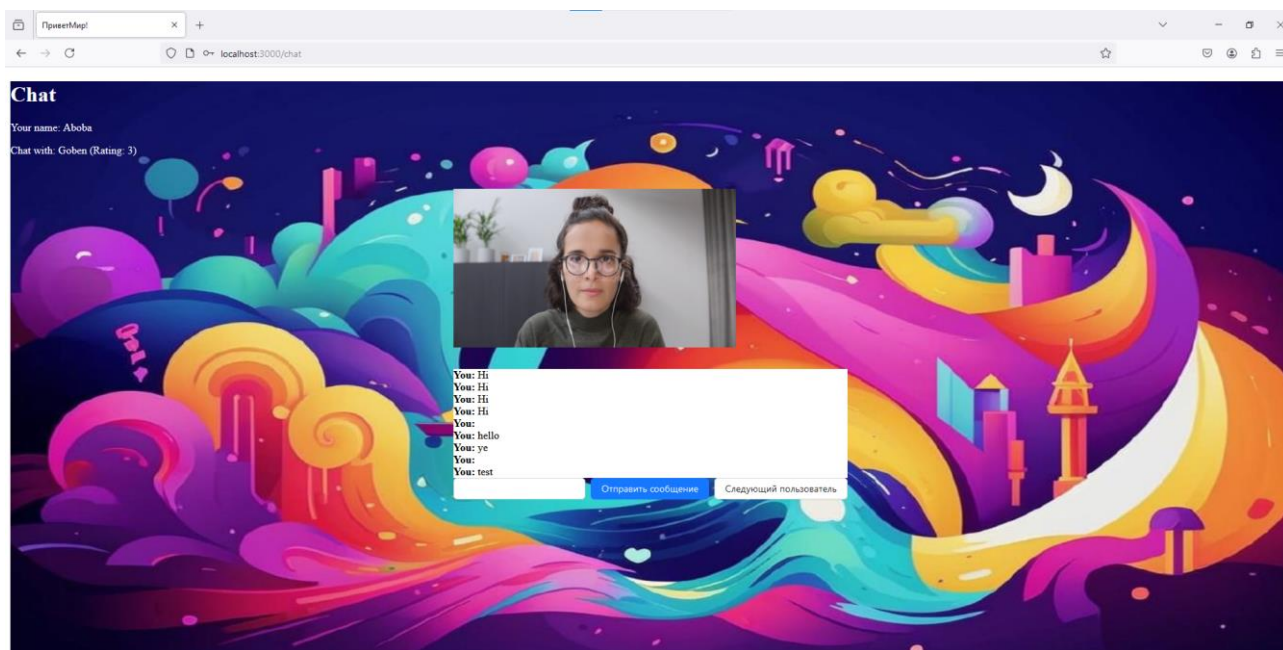


Рисунок Б.8 – Страница чата в браузере «Mozilla Firefox»