МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальность 1-40 05 01 Информационные системы
и технологии (по направлениям)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Программирование сетевых приложений»

на тему: «Многопользовательский чат с использованием *chatengine.io*»

Исполнитель: студентка гр. ИТП-41
Пимошенко А.С.
Руководитель: доцент
Курочка К.С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Мир переживает информационную революцию, вызванную широким внедрением в жизнь общества Интернета и Всемирной паутины, которая информационно связывает все сферы деятельности, все организации и конкретных людей. Суть этой революции заключается в интеграции в едином мировом информационном пространстве программно-аппаратных средств, средств связи и телекоммуникаций, информационных ресурсов в единую информационную инфраструктуру. *World Wide Web* (*WWW*) делает Интернет простым в использовании и обеспечивает ему мультимедийные возможности. Решающее значение для информационных стратегий организаций является присутствие в Интернете и создание распределенных информационных систем, обеспечивающих взаимодействие с заказчиками и поставщиками, маркетинг и многие другие виды деятельности. Все это вызывает повышенный интерес к технологиям создания инфокоммуникационных систем, основанных на распределенных вычислениях [1].

С течением времени многопользовательские чаты обретают все большую популярность. Однако в связи с большим количеством разновидностей, чаты имеют разный функционал и недостатки, например, отсутствие анонимности, отображение личных данных, нестабильная работа, отсутствие функции отправки изображений. В связи с этим задача создания многопользовательского чата с наличием широкого функционала, а также возможностью сокрытия личных данных является актуальной.

1 ОБЗОР СРЕДСТВ И ПОДХОДОВ СОЗДАНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКОГО ЧАТА. ОБЗОР АНАЛОГОВ

1.1 Основные понятия чата. История возникновения

Чат представляет собой программу или сервис для моментального обмена сообщениями. В некоторых случаях, чат размещается на странице какого-либо ресурса в виде отдельного окна. Его главное отличие от других подобных сервисов для общения, например, от форума, является то, что общение происходит без задержки. Пользователь отправляет сообщение другому человеку и практически сразу же получает на него ответ.

Кроме интернет-чатов, популярностью пользуются чаты в виде отдельного программного обеспечения. Яркими примерами подобных программ является *ICQ*. Обмен сообщениями в *Skype*, обладает теми же функциями.

Во второй половине XX века начали бурно развиваться компьютеры. Однако долгое время они были большими и слишком дорогими, что препятствовало тому, чтобы расходовать драгоценное машинное время на забавы с обменом сообщениями вместо расчётов атомных бомб. К тому же до конца 60-х годов они не были связаны друг с другом. Предок Интернета, сеть *ARPANET*, в 1969 году насчитывала только четыре связанных друг с другом научных компьютера. Чуть позже, в 1971 году, была придумана электронная почта, которая стала необычайно популярна ввиду своего удобства. Постепенно появились новые службы сообщений, такие, как списки почтовой рассылки, новостные группы и доски объявлений. Однако в то время сеть *ARPANET* ещё не могла легко взаимодействовать с другими сетями, построенными на других технических стандартах, что затрудняло её распространение. Но тем не менее эта проблема вскоре была решена после перехода сетей на протокол обмена данными *TCP/IP*, который успешно применяется до сих пор. Именно в 1983 году термин «Интернет» закрепился за сетью *ARPANET*. Программы для обмена текстовыми строками, несмотря на простоту самой идеи, появились не сразу. Примерно в 1974 году для мэйнфрейма *PLATO* была разработана программа *Talkomatic*, потенциально позволявшая общаться между тысячей терминалов системы. В 1980-х появилась система *Freelancing' Round table*. Однако по-настоящему популярным стал разработанный в 1988 году протокол, названный *Internet Relay Chat* (*IRC*), что примерно можно перевести как ретранслируемый интернет-разговор. Примерно в это же время появилось и распространилось само понятие «чат». Общение в *IRC* быстро стало популярным из-за простоты процесса и дружественности среды. В 1991 году во время операции «Буря в пустыне» была организована *IRC*-трансляция новостей – сообщения

со всего мира собирались в одном месте и в режиме реального времени передавались в *IRC*. Есть сведения, что подобным образом *IRC* использовался и во время путча в СССР, когда пользователи из Москвы моментально сообщали всему миру о происходящем на улицах. Для клиентов *IRC* написано множество ботов, например, *Eggdrop*, автоматизирующие многие рутинные операции. Самым известным из клиентов *IRC* стал *mIRC*; благодаря простой и эффективной системе команд для него было написано множество скриптов, которые также позволяют выполнять широкий спектр действий. Боты и *mIRC*-боты используются для различных игр в каналах – «Мафия», «Викторина» и других.

В настоящее время интерес к чатам падает. Их место заняли социальные сети, с их намного большими функциональностями. Популярность набирают только видеочаты [2].

1.2 Виды чатов

Веб-чат. Веб-чаты распространились в 90-х годах XX века. В некоторых случаях под чатом подразумевают веб-чат. Веб-чаты основаны на технологиях интернета (*HTTP* и *HTML*). Первоначально веб-чаты представляли собой страницу с разговором и форму ввода, посредством которой введённый текст отсылался на сервер. В подобных чатах сервер добавлял новые сообщения в текстовую область, удалял старые сообщения, обновлял файл. Чат осуществлялся с задержкой: веб-средства не позволяли серверу сообщить клиенту об изменениях – клиент мог только запрашивать данные сам с некоторой периодичностью. Это задержки были устранены с помощью технологий *AJAX* и *Flash*. Также существовали некоторые другие системы, не имевшие подобных недостатков.

Веб-чаты использовались для атак на пользователей. Этому способствовали уязвимости в программном обеспечении (скриптах). Поэтому многие веб-сервера, где находятся чаты, были вынуждены защищаться от атак.

Видеочат. Видеочаты – это обмен текстовыми сообщениями и трансляция изображений с веб-камер. Поначалу это были не видео-, а скорее, фоточаты: из-за низкой пропускной способности каналов отправлялся не видеопоток, а картинка с некоторыми интервалами, что, однако, давало возможность достаточно оперативно наблюдать смену эмоций у собеседника и было значительным прорывом. Позднее, конечно, стал транслироваться видеопоток, хотя и с низким разрешением. Веб-камеры являются простыми и дешёвыми, хотя обратная сторона этого – низкое разрешение видео и его плохое качество. Изображение получается с плохой цветопередачей, зашумлённое. Однако для целей общения такого качества более чем достаточно.

Голосовые чаты тоже явились развитием идей обмена сообщениями. В настоящее время в компьютерных играх широко применяется система *Discord*, позволяющая общаться голосом между членами команды, не отвлекаясь от управления игрой. А общение по *Skype* больше напоминает разговор по телефону, чем чат, хотя возможность отправки обычных текстовых сообщений в нём тоже присутствует.

Системы мгновенных сообщений или мессенджер. Это службы мгновенных сообщений (*Instant Messaging Service, IMS*), программы онлайн-консультанты (*OnlineSaler*) и программы-клиенты (*Instant Messenger, IM*) для обмена сообщениями в реальном времени через Интернет. Могут передаваться текстовые сообщения, звуковые сигналы, изображения, видео, а также производиться такие действия, как совместное рисование или игры. Многие из таких программ-клиентов могут применяться для организации групповых текстовых чатов или видеоконференций. Большинство *IM*-клиентов позволяет видеть, подключены ли в данный момент абоненты, занесённые в список контактов. В ранних версиях программ всё, что печатал пользователь, тут же передавалось. Если он делал ошибку и исправлял её, это тоже было видно. В таком режиме общение напоминало телефонный разговор. В современных программах сообщения появляются на мониторе собеседника уже после окончания редактирования и отправки сообщения.

Как правило, мессенджеры не работают самостоятельно, а подключаются к центральному компьютеру сети обмена сообщениями, называемому сервером. Поэтому мессенджеры и называют клиентами (клиентскими программами). Термин является понятием из клиент-серверных технологий.

Теле чаты. Используются на телеканалах, таких, как *MTV, RU.TV, Bridge-TV*. Сообщение передаётся путём отправки *SMS* с мобильного. Чаще всего это объявления о знакомствах или поздравления с праздниками. Также на некоторых каналах ведётся общение с диджеем или ведущим. Однако большинство сообщений платные [2].

1.3 Программная реализация чатов

Существует несколько разновидностей программной реализации чатов:

– *HTTP* или веб-чаты. Такой чат выглядит как обычная веб-страница, где можно прочесть последние несколько десятков фраз, написанные участниками чата и модераторами. Страница чата автоматически обновляется с заданной периодичностью;

– чаты, использующие технологию *Adobe Flash*. Вместо периодической перезагрузки страницы между клиентом и сервером открывается сокет, что

позволяет моментально отправлять или получать сообщения, расходуя меньше трафика;

– *IRC*, специализированный протокол для чатов. *IRC* (англ. *Internet Relay Chat*) – протокол прикладного уровня для обмена сообщениями в режиме реального времени. Разработан в основном для группового общения, также позволяет общаться через личные сообщения и обмениваться данными, в том числе файлами. *IRC* использует транспортный протокол *TCP* и криптографический *TLS* (опционально);

– программы-чаты для общения в локальных сетях (например, *Vypress Chat, Intranet Chat, Pichat*). Часто есть возможность передачи файлов;

– чаты, реализованные поверх сторонних протоколов (например, чат, использующий *ICQ*);

– чаты, работающие по схеме клиент-сервер, это позволяет использовать их в сетях со сложной конфигурацией, а также управлять клиентскими приложениями (например, *Mychat, Jabber*);

–чаты, работающие в одноранговых сетях. У них нет потребности в отдельном сервере, они часто используют возможности технологий *DHT* и *TCP Relay* (пример: *Tox*);

– чаты, использующие технологию *Push*. Вместо периодической отправки запросов серверу о новых сообщениях, используются входящие сообщения от сервера, что позволяет отправлять и получать сообщения, расходуя меньше трафика (например, *WinGeoChat*);

– полностью анонимные чаты. В них собеседник не предполагает с кем общается и при каждом новом соединении общается с новым человеком [3].

1.5 Основы сетевого взаимодействия

В начале 80-х годов международная организация по стандартизации (*International Standardization Organization – ISO*) разработала модель *OSI*, которая сыграла значительную роль в развитии сетей. Эталонная модель *OSI*, иногда называемая стеком *OSI* представляет собой 7-уровневую сетевую иерархию. горизонтальную модель на базе протоколов, обеспечивающую механизм взаимодействия программ и процессов на различных машинах. Вертикальная модель базируется на основе услуг, обеспечиваемых соседними уровнями друг другу на одной машине. В горизонтальной модели двум программам требуется общий протокол для обмена данными. В вертикальной – соседние уровни обмениваются данными с использованием интерфейсов *API*.

В ситуации, когда приложение обращается с запросом к прикладному уровню, например, к файловой службе. На основании этого запроса программное обеспечение прикладного уровня формирует сообщение стандартного формата. Обычное сообщение состоит из заголовка и поля данных. Заголовок содержит служебную информацию, которую необходимо передать через сеть прикладному уровню машины-адресата, чтобы сообщить ему, какую работу надо выполнить. Поле данных сообщения может быть пустым или содержать какие-либо данные, которые необходимо записать в удаленный файл. Но для того чтобы доставить эту информацию по назначению, предстоит решить еще много задач, ответственность за которые несут нижележащие уровни.

После формирования сообщения прикладной уровень направляет его вниз по стеку представительному уровню. Протокол представительного уровня на основании информации, полученной из заголовка прикладного уровня, выполняет требуемые действия и добавляет к сообщению собственную служебную информацию – заголовок представительного уровня, в котором содержатся указания для протокола представительного уровня машины-адресата.

Полученное в результате сообщение передается вниз сеансовому уровню, который в свою очередь добавляет свой заголовок, и т. д.

Наконец, сообщение достигает нижнего, физического уровня, который собственно и передает его по линиям связи машине-адресату. К этому моменту сообщение «обрастает» заголовками всех уровней (рис. 1.1).

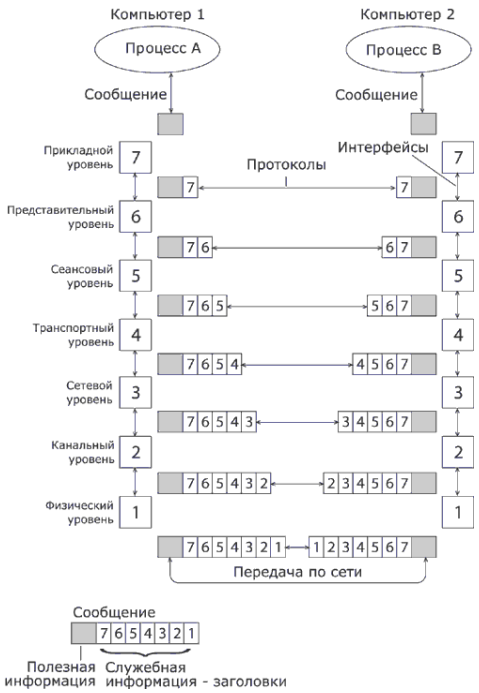

Рисунок 1.1 – Эталонная модель взаимодействия открытых систем (*ISO/0SI*).

Когда сообщение по сети поступает на машину-адресат, оно принимается ее физическим уровнем и последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует и обрабатывает заголовок своего уровня, выполняя соответствующие данному уровню функции, а затем удаляет этот заголовок и передает сообщение вышележащему уровню.

В модели *OSI* различаются два основных типа протоколов. В протоколах с установлением соединения перед обменом данными отправитель и получатель должны сначала установить соединение и, возможно, выбрать некоторые параметры протокола, которые они будут использовать при обмене данными. После завершения диалога они должны разорвать это соединение. Вторая группа протоколов – протоколы без предварительного установления соединения. Такие протоколы называются также дейтаграммными протоколами. Отправитель просто передает сообщение, когда оно готово. При взаимодействии компьютеров используются протоколы обоих типов.

Каждый уровень модели *OSI* имеет чёткие функции.

1.5.1 На физическом уровне модели *OSI* определяются следующие характеристики сетевых компонентов:

– типы соединений сред передачи данных, физические топологии сети, способы передачи данных (с цифровым или аналоговым кодированием сигналов), виды синхронизации передаваемых данных;

– разделение каналов связи с использованием частотного и временного мультиплексирования.

Физический уровень не включает описание среды передачи. Однако реализации протоколов физического уровня специфичны для конкретной среды передачи. С физическим уровнем обычно ассоциируется подключение сетевого оборудования.

1.5.2 Канальный уровень определяет логическую топологию сети, правила получения доступа к среде передачи данных, решает вопросы, связанные с адресацией физических устройств в рамках логической сети и управлением передачей информации (синхронизация передачи и сервис соединений) между сетевыми устройствами. Протоколы канального уровня реализуются для достижения следующих основных целей:

– организации битов физического уровня (двоичные единицы и нули) в логические группы информации, называемые фреймами или кадрами. Фрейм является единицей данных канального уровня, состоящей из непрерывной последовательности сгруппированных битов, имеющей заголовок и окончание;

– обнаружения (а иногда и исправления) ошибок при передаче;

– управления потоками данных (для устройств, работающих на этом уровне, например, мостов);

– идентификации компьютеров в сети по их физическим адресам.

1.5.3 Сетевой уровень служит для образования единой транспортной системы, объединяющей несколько сетей, причем эти сети могут использовать совершенно различные принципы передачи сообщений между конечными узлами и обладать произвольной структурой связей. Функции сетевого уровня достаточно разнообразны.

Сообщения сетевого уровня принято называть пакетами. При организации доставки пакетов на сетевом уровне используется понятие «номер сети». В этом случае адрес получателя состоит из старшей части - номера сети и младшей – номера узла в этой сети. Все узлы одной сети должны иметь одну и ту же старшую часть адреса

1.5.4 На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Хотя некоторые приложения имеют собственные средства обработки ошибок, существуют и такие, которые предпочитают сразу иметь дело с надежным соединением. Транспортный уровень обеспечивает приложениям или верхним уровням стека - прикладному и сеансовому - передачу данных с той степенью надежности, которая им требуется.

Модель *OSI* определяет пять классов сервиса, предоставляемых транспортным уровнем. Эти виды сервиса отличаются качеством предоставляемых услуг: срочностью, возможностью восстановления прерванной связи, наличием средств мультиплексирования нескольких соединений между различными прикладными протоколами через общий транспортный протокол, а главное – способностью к обнаружению и исправлению ошибок передачи, таких как искажение, потеря и дублирование пакетов [5].

1.5.5 Сеансовый уровень обеспечивает управление диалогом: фиксирует, какая из сторон является активной в настоящий момент, предоставляет средства синхронизации. Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, а не начинать все с начала. Сеансовый уровень реализует управление диалогом с использованием одного из трёх способов общения: симплекс, полудуплекс и полный дуплекс.

1.5.6 Представительный уровень имеет дело с формой представления передаваемой по сети информации, не меняя при этом ее содержания. За счет уровня представления информация, передаваемая прикладным уровнем одной системы, всегда понятна прикладному уровню другой системы. На этом уровне может выполняться шифрование и дешифрование данных, благодаря

которому секретность обмена данными обеспечивается сразу для всех прикладных служб. Примером такого протокола является протокол *Secure Socket Layer* (*SSL*), который обеспечивает секретный обмен сообщениями для протоколов прикладного уровня стека *TCP/IP*.

1.5.7 Прикладной уровень – это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые *Web*-страницы, а также организуют свою совместную работу, например, с помощью протокола электронной почты. Единица данных, которой оперирует прикладной уровень, обычно называется сообщением. Существует очень большое разнообразие служб прикладного уровня [6 с. 126]. К числу наиболее распространенных протоколов верхних уровней относятся:

– *FTP* – протокол переноса файлов;
– *TFTP* – упрощенный протокол переноса файлов;
– *SMTP* – простой протокол почтового обмена;
– *CMIP* – общий протокол управления информацией;
– *SNMP* – простой протокол управления сетью;
– *NFS* – сетевая файловая система.

1.6 Существующие аналоги

На сегодняшний день существует большое количество многопользовательских чатов, что обусловлено современными тенденциями и образом жизни. Все они чем-то отличаются, но в целом, имеют схожий функционал. Для более подробного анализа следует рассмотреть один из чатов, например, мессенджер *Viber*.

Данное приложение позволяет отправлять сообщения, сообщать видео- и голосовые звонки через Интернет. Голосовые вызовы между пользователями с установленным *Viber* бесплатны (оплачивается только интернет-трафик по тарифу оператора связи). *Viber* имеет возможность отправлять текстовые, голосовые и видеосообщения, документы, изображения, видеозаписи и файлы, а также в автономном режиме. Приемуществом является то, что в мессенджере предусмотрена функция шифрования, которая обеспечивает тайну переписка. Главным недостатком для авторизации пользователей и поиска контактов приложение использует номер телефона и передает содержимое телефонной адресной книги (имена и телефоны всех контактов) на серверы корпорации *Viber Media S.à r.l.*, Люксембург. Они же собирают информацию о совершенных звонках и переданных сообщениях, длительности звонков, участниках звонков и чатов.

1.7 Результат аналитического обзора

В результате аналитического обзора был выявлен ряд недостатков, главным недостатком является отсутствие функции сокрытия личной информации. В связи с этим можно сформулировать задачу, которую необходимо решить.

Требуется разработать сетевое приложение, реализующее многопользовательский чат на основе *chatengine.io*. Приложение должно иметь систему аутентификации, а также механизм как общих, так и личных сообщений. Предусмотреть возможность отправки файлов, удаления чатов, удаления и добавления пользователей в чат, а также предусмотреть анонимность.

Для решения задачи необходимо разработать приложение на высокоуровневом языке программирования. Также необходимо провести верификации работы разработанного приложения.

Программа должна иметь графический пользовательский интерфейс, содержащий список доступных чатов, кнопку создания чата, поле ввода сообщения и прикрепления файла, дополнительные кнопки для взаимодействия с чатами и пользователями. Результатом работы приложения является функционирующий чат, выполняющий все предусмотренные функции.

2 АРХИТЕКТУРА МНОГОПОЛЬЗОВАТЕЛЬСКОГО ЧАТА

2.1 Схема работы сервера многопользовательского чата

Библиотека *chatengine.io* является удобным и популярным инструментом для создания многопользовательских чатов. Для корректной работы данная библиотека использует веб-сокеты и *Rest API*.

Веб-сокеты (*Web Sockets*) – это передовая технология, которая позволяет создавать интерактивное соединение между клиентом (браузером) и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от *HTTP*, позволяют работать с двунаправленным потоком данных, что делает эту технологию совершенно уникальной.

У веб-сокетов также есть возможность шифровать передаваемые данные, для этого используется надстройка над протоколом – *WSS*. Если передаваемые данные не зашифрованы, они становятся объектом для привлечения таких угроз, как несанкционированный доступ к клиенту третьих сторон, использование вредоносного ПО. Специальные надстройки протоколов передачи данных кодируют информацию на стороне отправителя и раскодируют на стороне получателя, оставляя ее зашифрованной для любых посредников. Так достигается безопасный транспортный уровень. Веб-сокеты позволяют установить одно соединение, а сервер сам отправит новые сообщения, когда они появятся. Пример работы веб-сокетов представлен на рисунке 2.1.

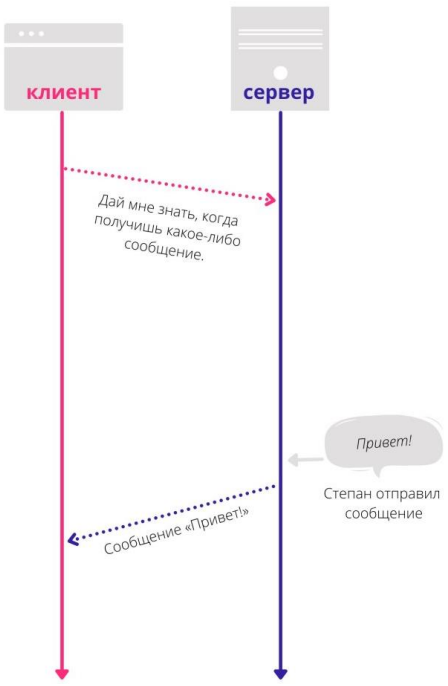

Рисунок 2.1 – Схема обмена сообщениями через протокол *WebSocket*

В интернете существуют строгие правила для передачи данных между клиентом и сервером (стек протоколов *TCP/IP*), но нет жестких правил о том, как устанавливать соединение и структурировать передаваемое сообщение. А это влияет на скорость.

Для установления соединения веб-сокет применяет метод открывающего рукопожатия. Он заключается в том, что клиент предваряет отправку/получение сообщений предварительным запросом, в котором клиент и сервер «договариваются» использовать веб-сокеты. Это и есть «рукопожатие». Структура такого запроса похожа на *HTTP*, но немного отличается от него. Затем клиент и сервер обмениваются данными уже в рамках этого соединения.

Кроме того, у веб-сокетов есть дополнительные расширения, которые дополняют и расширяют протокол. Например, есть расширение для сжатия данных. Или возможность передавать данные в формате протоколов *SOAP*, *WAMP* или *XMPP*. Чтобы эти расширения работали, они должны поддерживаться и сервером, и клиентом.

Rest API – это передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая *HTTP*-вызовы через интернет или любую другую сеть.

API – это набор инструментов, который позволяет одним программам работать с другими. *API* предусматривает, что программы могут работать в том числе и на разных компьютерах. В этом случае требуется организовать интерфейс *API* так, чтобы ПО могло запрашивать функции друг друга через сеть.

Также *API* должно учитывать, что программы могут быть написаны на различных языках программирования и работать в разных операционных системах.

В *API*-системе четыре классических метода:

– *GET* — метод чтения информации;
– *POST* — создание новых записей;
– *PUT* — редактирование записей;
– *DELETE* — удаление записей.

Чтобы избавиться от путаницы, когда на *API* происходит ошибка, нужно обрабатывать ошибки аккуратно и возвращать коды отклика *HTTP*, указывающие, какая именно ошибка произошла. Таким образом те, кто поддерживает *API*, получают достаточную информацию для понимания возникшей проблемы. Недопустимо, чтобы ошибки обваливали систему, поэтому и без обработки их оставлять нельзя, и заниматься такой обработкой должен потребитель *API*. Механизм работы *Rest API* изображен на рисунке 2.2.

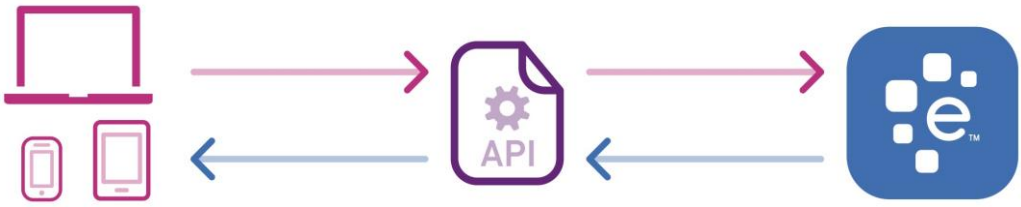

Рисунок 2.2 – Схема работы *Rest API*

Каждый *Rest API* запрос сообщает о результатах работы числовыми кодами — *HTTP*-статусами. Также *Rest API* позволяет обмениваться не только текстовой информацией. С помощью этого инструмента можно передавать файлы и данные в специальных форматах: *XML*, *JSON*, *Protobuf*.

2.2 Структура базы данных

Firebase – это облачная база данных, которая позволяет пользователям хранить и получать сохраненную информацию, а также имеет удобные средства и методы взаимодействия с ней.

Firebase хранит текстовые данные в *JSON* формате и предоставляет удобные методы для чтения, обновления и извлечения данных. Также, *Firebase* может помочь с регистрацией и авторизацией пользователей, хранением сессий (авторизованные пользователи), медиафайлов к которым с легкостью предоставляет доступ благодаря *Cloud Storage*.

База данных имеет ряд преимуществ. Обладает широким набором функций: создание и хранение базы данных, аутентификация, аналитика, хранение файлов. Благодаря тому, что сервисы находятся в облаке, программные продукты можно легко масштабировать. Также данная платформа является гибкой и универсальной.

В приложении *Firebase* используется для реализации регистрации и аутентификации пользователей.

Механизм взаимодействия базы данных с приложением изображен на рисунке 2.3.

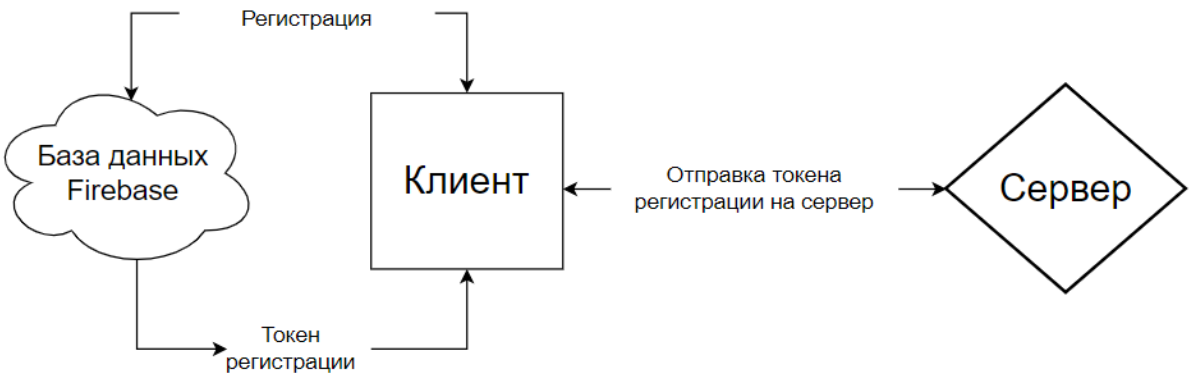

Рисунок 2.3 – Механизм работы облачной базы данных

Хранение данных пользователя, таких как имя и другая информация, также осуществляется в облачной базе данных.

2.3 Используемые библиотеки для реализации клиент-серверного приложения

Для написания программы используется библиотека *React.js*. *React* – *JavaScript*-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. *React* разрабатывается и поддерживается *Facebook*, *Instagram* и сообществом отдельных разработчиков и корпораций. *React* может использоваться для разработки одностраничных и мобильных приложений. Его цель – предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов *React* часто используется с другими библиотеками, такими как *MobX*, *Redux* и *GraphQL*.

Данная библиотека имеет ряд особенностей и преимуществ. Например, однонаправленная передача данных. Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через *callback*-функции. Такой механизм называют «свойства вниз, события наверх». Также *React* использует виртуальный *DOM*). *React* создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления *DOM* браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:

– *shouldComponentUpdate* позволяет предотвратить перерисовку компонента с помощью возврата *false*, если перерисовка не нужна;

– *componentDidMount* вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удаленного источника через *API*.

– *render* важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.

Хуки позволяют использовать состояние и другие возможности *React* без написания классов. Построение пользовательских хуков позволяет помещать логику компонента в повторно используемые функции.

Для выполнения *HTTP* запросов используется библиотека *axios*. *Axios* – это *JavaScript*-библиотека для выполнения либо *HTTP*-запросов в *Node.js*, либо *XMLHttpRequests* в браузере. Она поддерживает промисы – новинку *ES6*. Одна из особенностей, которая делает её лучше *fetch()* – автоматические преобразования *JSON*-данных.

2.4 Результат проектирования многопользовательского чата

Этапу написания любой программы, предшествует этап проектирования приложения. На этой стадии происходит анализ и выявление классов и их функциональности, что позволяет лучше понять их взаимодействие друг с другом. Поэтому, сначала следует спроектировать модель, основываясь на которой писаться сама программа.

В предыдущих главах была рассмотрена клиент-серверная архитектура, на основе которой и будет строиться всё приложение. Из этого следует, что в программе должны быть компоненты, отвечающие за подключение к базе данных, реализацию клиента и подключение к серверу, а также отвечающие за реализацию отправки и получения сообщений.

Для каждого подключения, серверу необходимо создать некий обработчик, который будет «общаться» непосредственно с клиентом и передавать именно те данные, которые требуется обработать.

Компонентами приложения являются:

– клиент;

– сервер;

– база данных.

Клиент отправляет запрос на сервер, сервер проверяет наличие пользователя в базе данных, при отсутствии информации о клиенте в базе данных, сервер создает пользователя с информацией о клиента, далее сервер отправляет ответ клиенту. Схема архитектуры приложения представлена на рисунке 2.4.

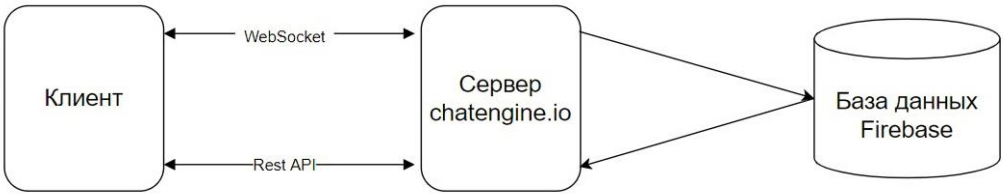

Рисунок 2.4 – Архитектура приложения

Реализация серверной части выполняется с помощью библиотеки *react-chat-engine*. В файле *index.js* хранятся все необходимые методы для подключения.

Для реализации механизма регистрации и аутентификации пользователя используется облачная база данных. Инициализация облачной базы данных описана в файле *Firebase.js*. Для реализации из соответствующей библиотеки импортируются необходимые методы, указываются все необходимые данные для подключения к базе данных.

Аутентификацию и регистрацию реализует файл *Login.js*. Данный файл содержит функцию, реализующую свои функции посредством регистрации и аутентификации пользователя с помощью *Google* почты. Проверка на наличие пользователя осуществляется в файле *Chats.js*. После обращения к серверу, в случае, если указанный пользователь не найден, создается новый пользователь.

Chats.js содержит в себе экспортируемые элементы. Библиотеку *axios*, которая упрощает работу с запросами на сервер, методы из файла для аутентификации.

Функцию отображения всех взаимодействующих элементов на странице выполняет *App.js*. Для этого необходимо импортировать все созданные компоненты, такие как *Login.js*, *Chats.js*, *AuthContext.js*, а также некоторые компоненты из библиотеки *react-router-dom*. Далее *App.js* импортируется в файл *index.js*

Схема взаимодействия компонентов приложения представлена на рисунке 2.5.

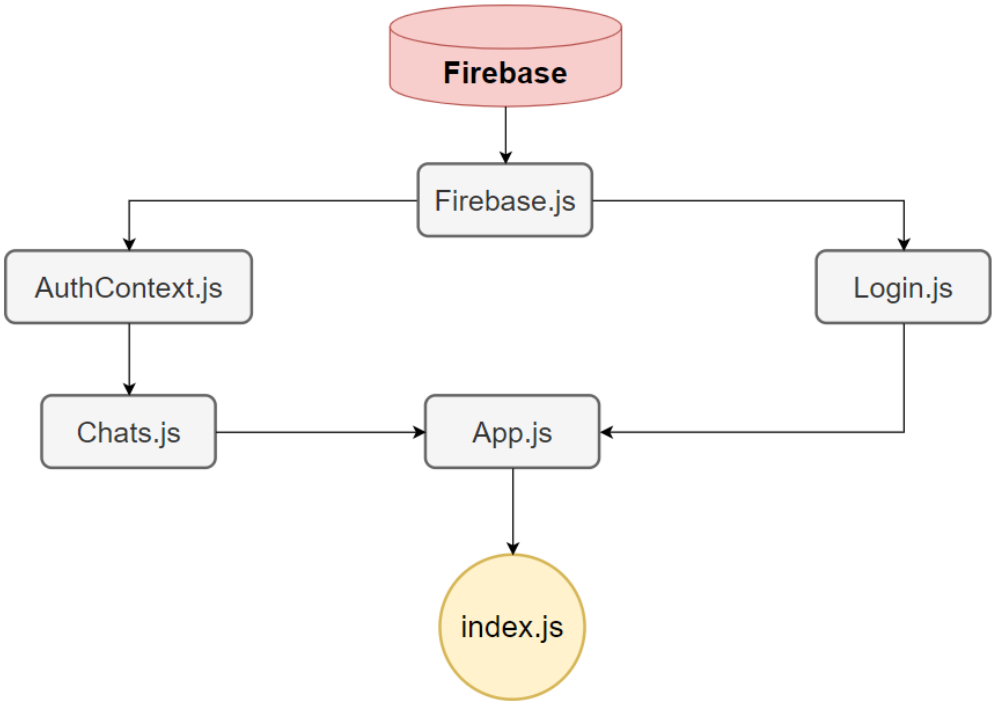

Рисунок 2.5 – Схема взаимодействия классов программы

На представленной выше диаграмме описано взаимодействие классов программы. Для определения полного функционала программы, необходимо более детально рассмотреть компоненты программы.

Класс *Firebase.js* экспортирует константу *auth*, включающую в себя значение, которое с помощью глобального пространства имен *firebase*, из которого осуществляется доступ ко всем сервисам, создает и инициализирует экземпляр приложения *Firebase*.

Класс *AuthContex.js* содержит функцию *useAuth()*, которая принимает объект контекста и возвращает текущее значение контекста. Функция *AuthProvider()* отвечает за состояние, и в необходимом случает обновляет его.

Класс *Login.js* содержит функцию *Login()*, которая содержит в себе функцию *signInWithGoogle()*, осуществляющую аутентификацию и регистрацию через *Google* почту. Функция *Login()* возвращает форму входа.

Класс *Chats.js* содержит функцию *Chats()*, которая возвращает окно многопользовательского чата. Функция *Chats()* использует импортированные методы из других классов, осуществляет взаимодействие пользователя с графическим интерфейсом. Асинхронная функция *getFile()* позволяет получать отправленные пользователем картинки. Также с помощью запроса функция получает заголовки, принадлежащие пользователю, если их нет, создает нового пользователя и передает его данные на сервер.

Приложение работает синхронно, то есть при выполнении операции программа ждет и не переходит к выполнению следующего действия. Если программа находится в состоянии ожидания, то вызов *API* не завершался. Получив ответ, *API* передает ответ, что ведет за собой выполнение следующего процесса.

Синхронизация приложения выполнена с помощью асинхронных функций. Асинхронная функция *handleLogout()* отвечает за выход из аккаунта пользователя. При выходе из аккаунта информация о пользователе обновляется у всех. Также синхронизация происходит посредством функции *setSendingMessage()*. После отправки сообщения пользователем, все остальные пользователи получают отправленное сообщение.

Сетевые коммуникации происходят посредством *WebSocket*. *WebSocket* отправляет запрос на сервер, для получения ответа, если клиент не получил ответ, *WebSocket* снова отправляет запрос, и так до того момента, пока клиент не получит ответ от сервера.

За реализацию *Rest API* отвечают функции: *getOrCreateSession*() – создает авторизированную сессию для пользователя, *getChats*() – получает список доступных чатов, *newChat*() – создает новый чат, *getLatestChats*() – получение последнего чата, *getOrCreateChat*() – получение или создание чата, *editChat*() – изменение чата, *deleteChat*() – удаление чата, *addPerson*() – добавление пользователя в чат, *removePerson*() – удаление пользователя из чата, *leaveChat*() – выход пользователя из чата, *getMessages*() – получение сообщений, *getLatestMessages*() – получение последнего сообщения, *readMessage*() – чтение сообщений, *deleteMessage*() – удаление сообщений.

3 ВЕРИФИКАЦИЯ ПРИЛОЖЕНИЯ

3.1 Графический интерфейс программы

Для полноценного использования, в приложении имеется графический интерфейс, разработанный с помощью библиотеки *React.* Библиотека предоставляет возможность создать современный дизайн, который привлечет пользователя.

Графический интерфейс приложения представлен на рисунке 3.1.

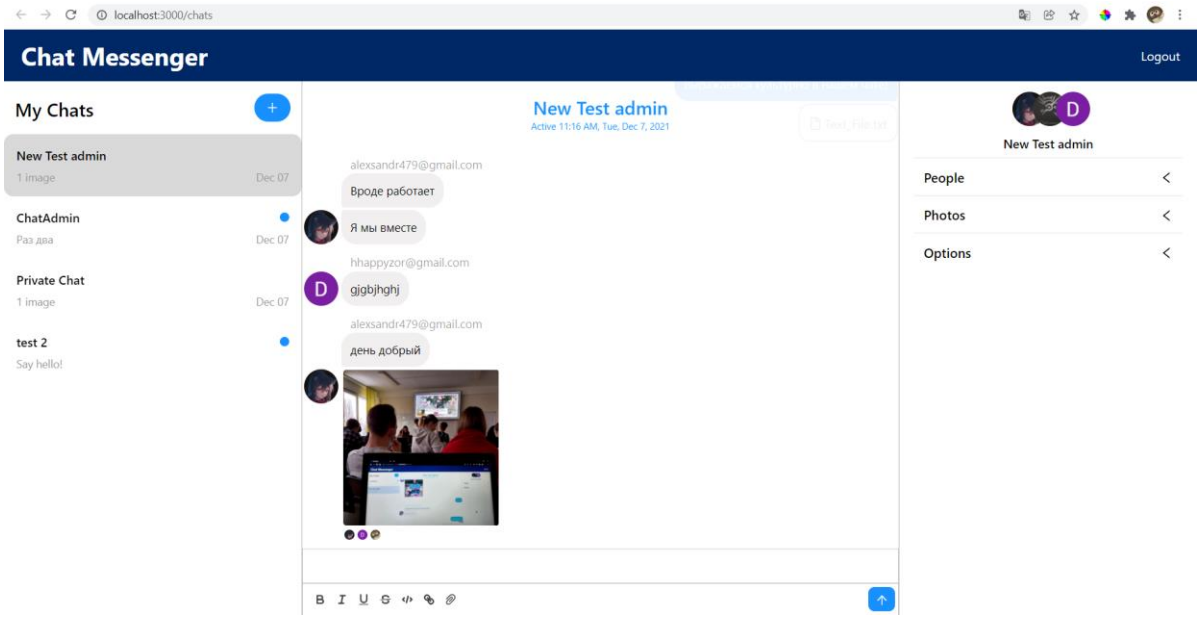

Рисунок 3.1 – Графический интерфейс приложения

Для входа и полноценного взаимодействия с приложением пользователю необходимо пройти этап регистрации на странице, отображенной после запуска приложения. Для этого в поле регистрации необходимо нажать на кнопку *«Sign In With Google»*. По нажатию этой кнопки для пользователя откроется новая вкладка браузера, в этой форме пользователю необходимо произвести вход с помощью электронной почты *Google*. После регистрации появляется надпись *«Loading…»*, что говорит о загрузке, после загрузки появляется окно приложения. Форма регистрации изображена на рисунке 3.2.

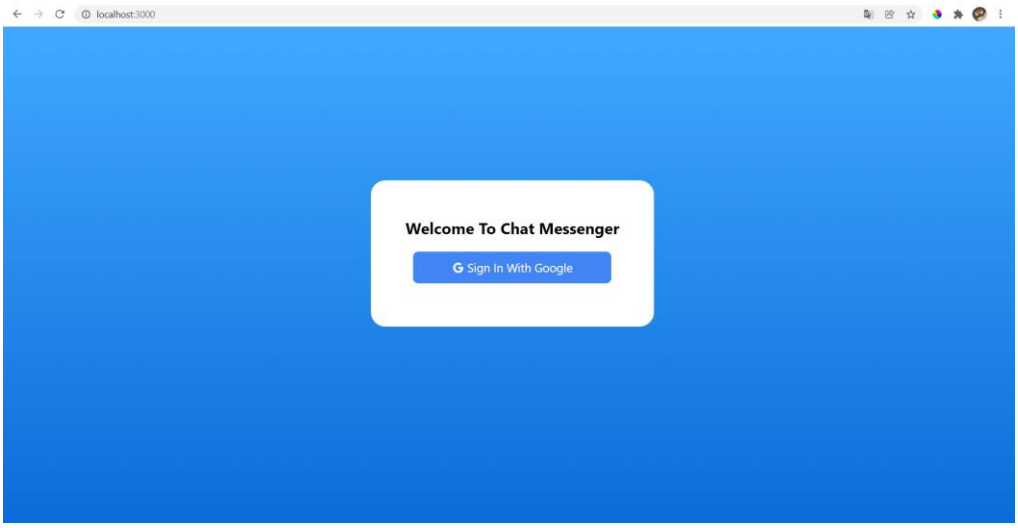

Рисунок 3.2 – Форма регистрации пользователя

Также предусмотрена кнопка выхода из аккаунта пользователя. При повторном входе пользователю не нужно регистрироваться, а по нажатию кнопки *«Sign In With Google»* произойдет аутентификация и пользователь сможет продолжить ранее законченный сеанс. Внешний вид кнопки выхода представлен на рисунке 3.3.

Рисунок 3.3 – Конопка выхода из аккаунта пользователя

После входа пользователю доступен весь функционал приложения. В левой части экрана находится элемент, отображающий все доступные пользователю чаты, как созданные пользователем, так и те, в которые пользователь был добавлен. Также в этой части находится кнопка «+» для создания нового чата. По нажатию кнопки появляется поле, в которое необходимо ввести название чата, который нужно создать. После ввода названия пользователю необходимо нажать клавишу *«Enter»*. По нажатию клавиши чат добавится в список чатов. Если в одном из доступных пользователю чатов появляется новое непрочитанное сообщение, этот чат помечается идентификатором в виде голубого круга. Данная возможность приложения упрощает работу с ним. Внешний вид списка доступных чатов представлен на рисунке 3.4, форма для ввода названия нового чата изображена на рисунке 3.5.

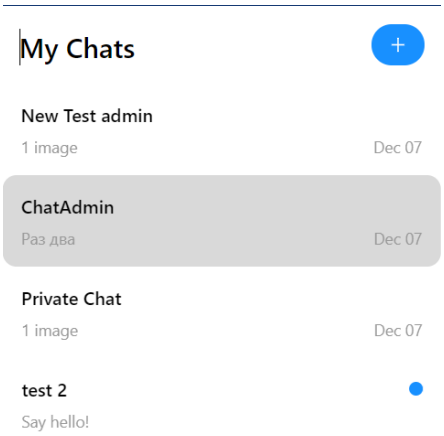

Рисунок 3.4 – Список доступных чатов

Рисунок 3.5 – Форма для ввода названия чата

В правой части экрана находится элемент, отображающий информацию об открытом чате. В верхней части панели находится название открытого чата, а также миниатюры фотографий участников чата. Ниже находятся три выпадающих списка *«People»*, *«Photos»*, *«Options»*.

При раскрытии списка *«People»* пользователь видит список участников чата, с миниатюрами их фотографий и именами, по наведению на участника чата создатель чата видит появляющуюся кнопку, по нажатию на которую можно удалить участника чата. Также в данном списке есть поле *«Type a username»*, которое осуществляет поиск участников для добавления. Чтобы добавить нового участника в чат, не обходимо ввести полностью или частично его имя, выбрать из результатов поиска необходимого пользователя и нажать на него, после чего пользователь будет добавлен в чат.

При раскрытии выпадающего списка *«Photos»* пользователь видит все вложения, ранее отправленные в чат, например, фотографии.

При раскрытии выпадающего списка *«Options»* пользователь видит кнопку *«Delete»*. По нажатию на которую пользователь может удалить открытый чат.

Функции удаления участника чата и удаления чата доступны только создателю чата.

Внешний вышеописанного элемента, а также содержимое выпадающих списков представлено на рисунке 3.6.

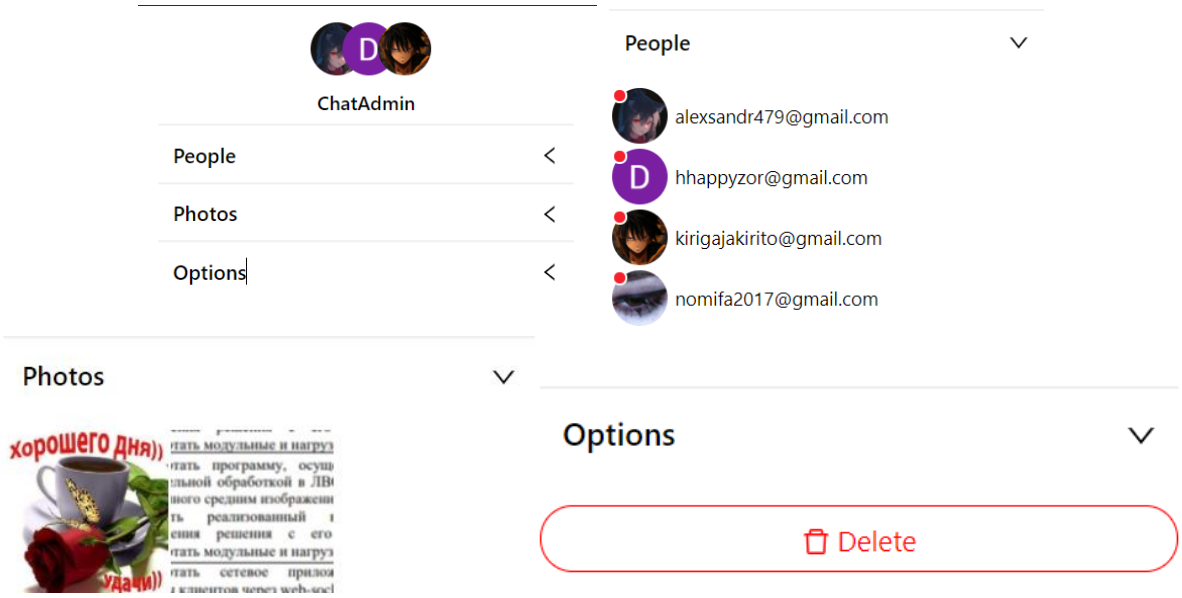

Рисунок 3.6 – Информация о чате

В центральной части экрана расположен элемент, отображающий все сообщения чата, как входящие, так и исходящие. Исходящие сообщения подсвечиваются синим цветом. Над каждым сообщением написано имя отправителя, а слева от сообщения отображена миниатюра фотографии сообщения. Также предусмотрено отображение информации о том, какой пользователь в данный момент набирает сообщение. В верхней части элемента находится название чата и дата его создания. Внизу находится поле для ввода сообщений, также есть возможность стилизации текста. Для того, чтобы отправить сообщение, пользователь должен ввести его в соответствующее поле и нажать на кнопку в виде стрелки. После проделанных действий сообщение появится в окне отображения сообщений. Для того, чтобы прикрепить файл, необходимо нажать на кнопку в виде скрепки. По нажатию кнопки откроется окно для выбора файла из локального хранилища, необходимо выбрать файл, подтвердить выбор и нажатием кнопки в виде стрелки отправить сообщение. Внешний вид центрального элемента приложения представлен на рисунке 3.7.

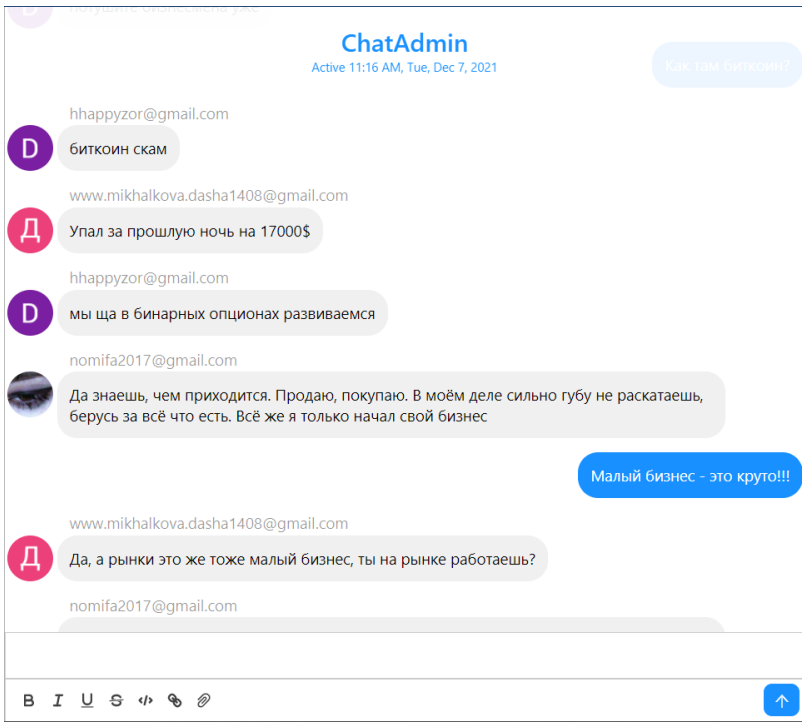

Рисунок 3.7 – Элемент отображения и отправки сообщений

Пользовательский интерфейс приложения очень прост и многофункционален. Благодаря этому у пользователя не возникнет проблем при взаимодействии с приложением. Пользовательский интерфейс информирует пользователя о всех происходящих процессов, что не вводит пользователя в заблуждение при ожидании загрузки или переподключении к серверу, также пользователь видит сообщения при возникновении ошибок.

3.2 Результаты тестирования многопользовательского чата

В ходе верификации необходимо проверить работоспособность приложения. То есть проверить все кнопки на выполнение ими необходимых действий. Для проверки функции создания чата необходимо нажать на кнопку «+» в списке чатов. По нажатию кнопки отображается поле для ввода названия нового чата. После ввода названия, по нажатию клавиши *Enter*, новый чат добавляется в список доступных чатов. Результат создания чата «*Chat for testing*» представлен на рисунке 3.8.

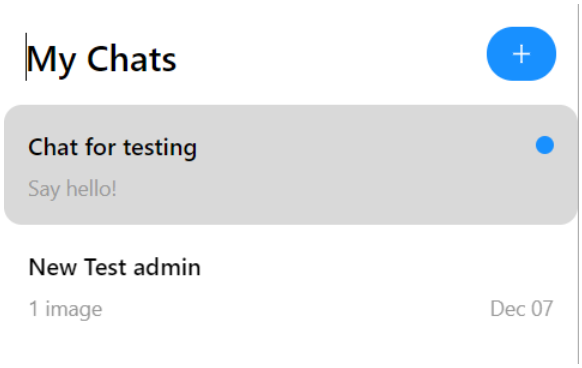

Рисунок 3.8 – Результат создания нового чата

Для проверки функции удаления пользователя из чата и удаления самого чата необходимо нажать на соответствующие кнопки. По нажатию кнопок пользователь будет удален из списка пользователей и больше не будет иметь доступ к чату, владельца чат нельзя удалить из чата. По нажатию кнопки для удаления чата, чат будет удален и будет недоступен. Результат удаления пользователя из чата и удаления чата «*Chat for testing*» представлен на рисунке 3.9.

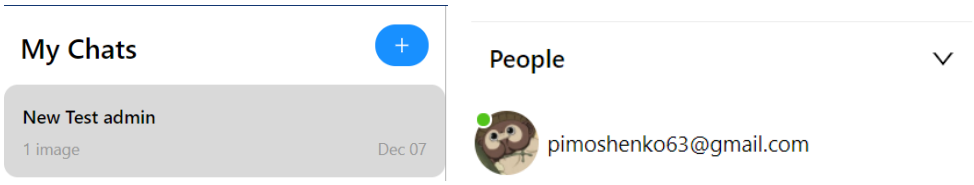

Рисунок 3.9 – Результат удаления чата и пользователя из чата

Для проверки функционала отправки сообщения необходимо перейти в чат, в поле, предназначенном для ввода сообщений, ввести необходимое для отправки сообщение и нажать кнопку для отправки сообщения. По нажатию кнопки сообщение появится в окне отображения сообщений с соответствующей датой и временем отправки. Для других пользователей будет отображено имя отправителя. Результат отправки и получения сообщения представлен на рисунке 3.10.

Рисунок 3.10 – Получение и отправка сообщений

Для отправки изображения необходимо в поле ввода сообщения нажать на кнопку прикрепления изображения, выбрать необходимое для отправки изображение из открытого окна директорий и нажать на кнопку отправки сообщения. Получения изображения происходит аналогично получению сообщений, над отправленным изображением отображается имя отправителя. Результат отправки и получения изображения представлен на рисунке 3.11.

Рисунок 3.11 – Отправка и получение сообщения

В ходе верификации также были проверены остальные функции многопользовательского чата, такие как добавление пользователя в чат, а также функционал ограничения прав. Пользователь, добавленный в чат, не имеет доступ к функционалу управления чатом, то есть удалению пользователей, добавлению пользователей и удалению чата. Данный функционал доступен только создателю чата.

Для проверки работоспособности были также выполнены нагрузочные тесты. Нагрузочное тестирование выполнялось с помощью специализированного ресурса *LoadStorm*. Данный ресурс позволяет провести тестирование приложения создавая ситуацию одновременного нахождения в приложении 50 пользователей. Для осуществления тестирования необходимо зарегистрироваться на сайте ресурса и создать тест, в котором указать адрес многопользовательского чата. Перед началом тестирования необходимо создать сценарий, который описывает действия пользователей. Время тестирования – примерно 30 минут, в это время 50 виртуальных пользователей совершают действия, заменяя реальных пользователей.

Результатом тестирования являются графики, отображающие работоспособность многопользовательского чата. На первом графике отображена информация о пропускной способности(*Throughput*), пользовательской нагрузке

(*User load*) и количестве запросов в секунду(*Requests per seconds*). Внешний вид графика представлен на рисунке 3.12.

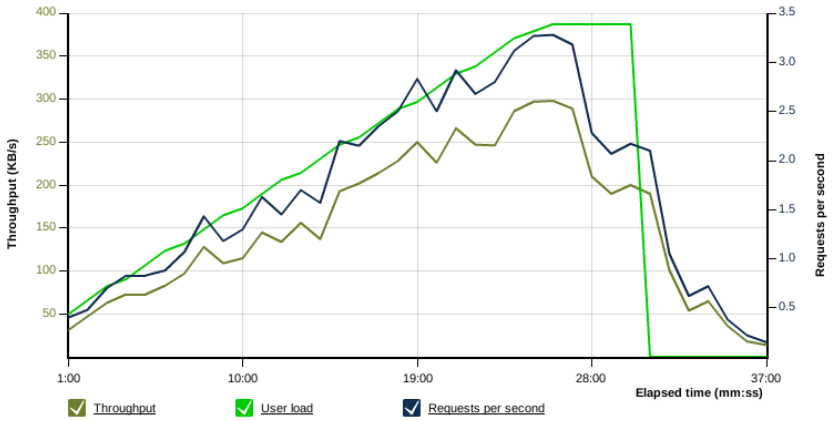

Рисунок 3.12 – Результат нагрузочных тестов

Второй график содержит информацию о времени отклика (*Response time*). Также на графике отображено среднее значение времени отклика(*Response time, average*), пиковое значение отклика(*Response time, peak*) и частота ошибок(*Errors rate*) в процентах. Среднее время отклика составило 0,281 секунды, пиковое значение отклика – 5,573 секунд, а частота ошибок составила 0%. График времени отклика изображен на рисунке 3.13.

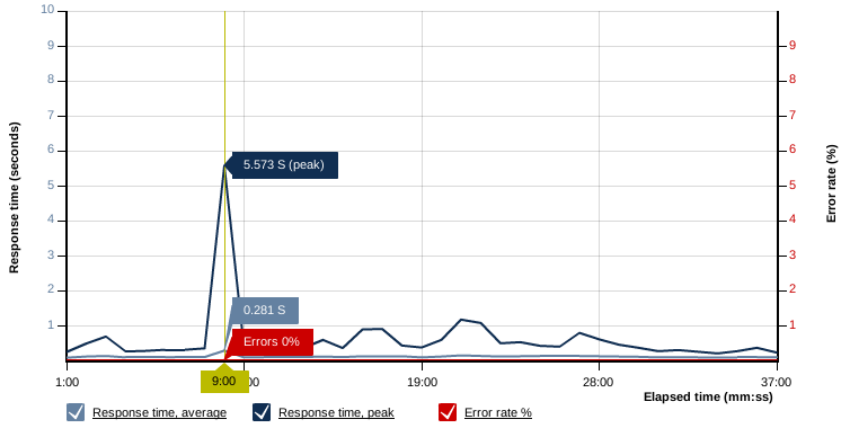

Рисунок 3.13 – Результат нагрузочных тестов

Для того, чтобы убедиться в реальной работоспособности программы, было проведено нагрузочное тестирование с реальными пользователями. Для этого 10 пользователей зашли на сайт приложения и в течение 30 минут взаимодействовали друг с другом. Сбои в работе приложения не обнаружены.

ЗАКЛЮЧЕНИЕ

В результате разработки курсового проекта были получены теоретические знания, а также получены практические навыки создания сетевых приложений. Были изучены базовые концепции построения сетевых приложений, позволяющих объединять несколько отдельных устройств для совместной работы и коммуникации пользователей.

В процессе выполнения изучены такие темы как: клиент-серверная архитектура сетевого приложения, протоколы транспортного уровня, средства высокоуровневых языков программирования для построения клиент-серверных приложений, а также соответствующие библиотеки.

Результатом разработки курсового проекта, является многопользовательский чат, написанный на языке программирования *JavaScript* с графическим интерфейсом пользователя с использованием библиотеки *React*. Приложение имеет широкий функционал, выполняющий все поставленные задачи. А также имеет простой и удобный интерфейс, который не вызовет сложности даже у неопытного пользователя.

Данное приложение решает ряд проблем. Например, помогает коммуницировать пользователям на расстоянии, когда другой вариант общения невозможен, обмениваться файлами. Приложение работает стабильно, что позволяет использовать его большому количеству пользователей.

Работа была проверена в системе «Антиплагиат», процент уникальности составил 80.61%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубаков – СПб: НИУ ИТМО, 2017. – 248 с.

2. Что такое чат: разновидности и история возникновения. – Электрон. данные. – Режим доступа: https://ktonanovenkogo.ru/voprosy-i-otvety/chat-chto-ehto-takoe.html. – Дата доступа: 07.12.2021.

3. Введение в сетевое программирование. – Электрон. данные – Режим доступа: – https://russianblogs.com/article/6258464747/ – Дата доступа: 07.12.2021.

4. Столлингс, В. Компьютерные сети, протоколы и технологии Интернета / В. Столлингс. - СПб.: BHV, 2005. - 832 с.

5. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов / В.Г. Олифер, Н.А. Олифер. – 4-е. изд. – СПб: Питер, 2007. – 960 с.

6. Руденков Н.А., Основы сетевых технологий: Учебник для вузов / Н.А. Руденков, Л.И. Долинер. – Екатеринбург : Изд-во Уральского Федерального ун-та им. Б.Н. Ельцина, 2011. – 342 с.

ПРИЛОЖЕНИЕ А
(обязательное)

Листинг программного комплекса

Firebase.js

```
//Конфигурация базы данных
import firebase from "firebase/app";
import 'firebase/auth';

export const auth = firebase.initializeApp({
   apiKey: "AIzaSyBjrjz_xDSwdh6QoZF5wOTgbUAfgIjdKfU",
   authDomain: "chat-77a8c.firebaseapp.com",
   projectId: "chat-77a8c",
   storageBucket: "chat-77a8c.appspot.com",
   messagingSenderId: "758264556723",
   appId: "1:758264556723:web:d5b641ace3135d7acb14c6"
}).auth();
```

AuthContext.js

```
//создание контекста аутентификации
import React, {useState,useEffect,useContext} from 'react';
import { auth } from '../Firebase/Firebase';
import { useHistory } from 'react-router';

const AuthContext = React.createContext();

export const useAuth = () => {
 const context = useContext(AuthContext);
 if (!context) {
    throw Error("Error in AuthContext")
 }
 return context;
}


export default function AuthProvider ({ children }) {
   //изменение состояния при авторизации
   const [loading, setLoading] = useState(true);
   const [user, setUser] = useState(null);
   const history = useHistory();

   useEffect(() => {
      auth.onAuthStateChanged((user) => {
         setUser(user);
         setLoading(false);
         if (user) {
```

```
            history.push('/chats')
          }
        })

    }, [user,history])
    //переменная, содержащая пользователя
    const value = { user };

    return (
        <AuthContext.Provider value={value}>
            {!loading && children }
        </AuthContext.Provider>
    )
}
```

Login.js

```
//реализация аутентификации и регистрации
import React from 'react';
import {GoogleOutlined} from '@ant-design/icons';
import firebase from 'firebase/app';
import { auth } from '../Firebase/Firebase';

export default function Login() {
//функция, реализующая вход и регистрацию с помощью Google почты
    function signInWithGoogle() {
        auth.signInWithRedirect(new firebase.auth.GoogleAuthProvider())
    }

    return (
        <div id='login-page'>
            <div id='login-card'>
                <h2>Welcome To Chat Messenger</h2>

                    <button onClick={signInWithGoogle} className='login-button google'><GoogleOutlined/>
Sign In With Google</button>
            </div>
        </div>
    )
}
```

Chats.js

```
//реализация чата
import React, {useState,useEffect} from 'react';
import { auth } from '../Firebase/Firebase';
import { ChatEngine } from 'react-chat-engine';
import { useHistory } from 'react-router';
import {useAuth} from '../contexts/AuthContext';
import axios from 'axios';
```

```
require('dotenv').config();



export default function Chats() {

    const {user} = useAuth();
    const [loading, setLoading] = useState(true);
    const history = useHistory();

    async function handleLogout() {
        await auth.signOut();
        history.push('/')
    }
    //функция получения картинок
    const getFile = async (url) => {
        const response = await fetch(url);
        const data = await response.blob();

        return new File([data], "userPhoto.jpg", { type: 'image/jpeg' } )
    }

    useEffect(() => {
        if (!user) {
            history.push('/');
            return;
        }
        //запрос на сервер, для получения пользователя
        axios.get('https://api.chatengine.io/users/me', {
            headers: {
                "project-id": process.env.REACT_APP_CHAT_ENGINE_ID,
                "user-name": user.email,
                "user-secret": user.uid
            }
        }).then( () => {
            setLoading(false);
        }).catch(() => {
            // если нет пользователя, то создать
            let formdata = new FormData();
            formdata.append("email", user.email);
            formdata.append("username", user.email);
            formdata.append("secret", user.uid);

        //получение аватара пользователя и его имени
        getFile(user.photoURL)
            .then((avatar) => {
                  formdata.append("avatar", avatar, avatar.name );

            axios.post('https://api.chatengine.io/users/',
```

```
                formdata,
                { headers: {
                   "private-key" : process.env.REACT_APP_CHAT_ENGINE_KEY
                }})
            .then(() => setLoading(false))
            .catch((error) => console.log(error))
            })
        })
    }, [user,history])

    if (!user || loading) {
        return "Loading ..."
    }
   //возврат компонента
    return (
        <div className='chats-page'>
            <div className="nav-bar">
                <div className="logo-tab">Chat Messenger</div>
                <div onClick={handleLogout} className='logout-tab'>Logout</div>
            </div>
            <ChatEngine
                height="calc(100vh - 66px)"
                projectID={process.env.REACT_APP_CHAT_ENGINE_ID}
                userName={user.email}
                userSecret={user.uid}
             />
        </div>
    )
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Руководство системного программиста

Общие сведения о программном обеспечении. Разработанное приложение является многопользовательским чатом. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор *Intel Core i3* с тактовой частотой 2,30 МГц или более;

– наличие клавиатуры, мыши и цветного монитора;

– операционная система *Windows 7* и выше;

– среда разработки *VisualStudio Code*;

– облачная база данных *Firebase;*

– 200 Мб оперативной памяти.

Структура приложения. Приложение включает в себя окно, которое имеет кнопки управления программы и связано с базой данных, данные которой представлены в виде таблиц.

Настройка приложения. Запуск приложения осуществляется путём запуска сервера в командной строке директории *Chat-App*, посредством команды *npm start*. Далее осуществляется переход по указанным ссылкам, в окне браузера открывается вкладка с приложением.

Для запуска приложения должен быть установлен *Node.js*.

ПРИЛОЖЕНИЕ В
(обязательное)

Руководство программиста

Назначение и условия применения программы. Разработанное приложение является многопользовательским чатом, осуществляющим функции получения и отправки сообщений, обмена изображениями. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор *Intel Core i3* с тактовой частотой 2,30 МГц или более;

– наличие клавиатуры, мыши и цветного монитора;

– операционная система *Windows 7* и выше;

– среда разработки *VisualStudio Code*;

– облачная база данных *Firebase;*

– 200 Мб оперативной памяти.

Характеристики приложения. Разработанное приложение написано на языке программирования *JavaScript* с помощью *React*. При запуске открывается окно программы с кнопками для управления приложением.

Обращение к приложению. Запуск приложения осуществляется путём запуска сервера в командной строке директории *Chat-App*, посредством команды *npm start*. Далее осуществляется переход по указанным ссылкам, в окне браузера открывается вкладка с приложением.

Для запуска приложения должен быть установлен *Node.js*.

ПРИЛОЖЕНИЕ Г
(обязательное)

Руководство пользователя

Введение. Разработанное приложение является многопользовательским чатом. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор *Intel Core i3* с тактовой частотой 2,30 МГц или более;

– наличие клавиатуры, мыши и цветного монитора;

– операционная система *Windows 7* и выше;

– среда разработки *VisualStudio Code*;

– облачная база данных *Firebase;*

– 200 Мб оперативной памяти.

Характеристики приложения. Разработанное приложение написано на языке программирования *JavaScript* с помощью *React*. При запуске открывается окно программы с кнопками для управления приложением.

Подготовка к работе. Запуск приложения осуществляется путём запуска сервера в командной строке директории *Chat-App*, посредством команды *npm start*.

Описание операций. После запуска приложения, перед пользователем открывается окно регистрации, по нажатию кнопки *«Sign In With Google»* происходит регистрация, если пользователь зарегистрирован – аутентификация. После входа пользователю доступен весь функционал приложения.

В левой части экрана находится список доступных чатов, открытие чата происходит по клику на необходимый чат, для того, чтобы создать новый чат необходимо нажать кнопку «+» и ввести название чата в соответствующее поле. По нажатию клавиши *Enter*, новый чат будет добавлен в список чатов.

В правой части экрана находится элемент, отображающий сведения о выбранном чате. Здесь находится название чата, список участников, список вложений, а для создателя чата – дополнительный функционал, в виде удаления пользователя из чата и удаления чата.

В центральной части экрана находится элемент, содержащий входящие и исходящие сообщения, а также поле для ввода сообщений, кнопку для отправки сообщений, кнопку для прикрепления, а также дополнительные кнопки для стилизации сообщений.

ПРИЛОЖЕНИЕ Д
(обязательное)

Схема архитектуры приложения формата А1