**СОДЕРЖАНИЕ**

[Введение 3](#_Toc132049495)

[1 Анализ данных и постановка задач на дипломное проектирование](#_Toc132049496) 5

[1.1 Анализ и описание функицональных возможностей программного средства](#_Toc132049497) 5

[1.2 Обзор существующих программных средств по теме дипломного проекта](#_Toc132049498) 9

[1.3 Обоснования и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек](#_Toc132049500) 18

[1.4 Постановка задача на дипломное проектированиеахах](#_Toc132049500) 25

[2 Проектирование, разработка и тестирование программного средства](#_Toc132049501) 26

[2.1 Проектирование архитектуры и описание состояний программного средства](#_Toc132049502) 26

[2.2 Формализация предметной области программного средства](#_Toc132049503) 35

[2.3 Проектирование и реализация способа хранения данных программног средства](#_Toc132049503) 36

[2.4 Проектирование и разработка графического интерфейса](#_Toc132049503) 41

[2.5 Описание и реализация используемых в программной среде алгоритмов](#_Toc132049503) 47

[2.6 Тестирование программного средства](#_Toc132049503) 54

[Заключение](#_Toc132049513)

Список использованных источников

[Приложение](#_Toc132049515) А (обязательное) [Листинг программы](#_Toc132049516)

**ВВЕДЕНИЕ**

В настоящие время приложения для социального взаимодействия резко шагнули вперед. Если в начале 2000 самым распространённым способом была простая сотовая связь, то с 2004 после релиза *Facebook* социальные сети стали резко набирать популярность [1].

Изначально *Facebook* использовали студенты и обычные люди для коммуникации между собой для повседневного общение. Огромный успех социальных сетей у людей привлек к этой технологии внимание предприятий, которые стали активно внедрять ее для внутреннего использования, чтобы повысить эффективность общения между сотрудниками, находить полезные знания неформальными способами и более целенаправленно доставлять нужную информацию.

В результате платформы для организации корпоративных социальных сете стали пользоваться повышенным спросом.Еще несколько лет назад работодатели не стали бы даже обсуждать перспективу внедрения корпоративной сети в их фирме, поскольку большинство из них полагало, а некоторые и сейчас считают, что сети − бессмысленная игрушка, на которую сотрудники тратят рабочее время. По данным исследования *Ipsos* для *Microsoft*, до сих пор в 34% отечественных компаний доступ к социальным сетям всех видов полностью запрещен. При этом около 40% сотрудников различных организаций убеждены, что социальные сети будут способствовать эффективному взаимодействию с коллегами [2].

Сейчас мы видим, что ситуация медленно, но верно меняется. Постепенно самые продвинутые работодатели начинают понимать, насколько удобно то, что сотрудники практически в любое время находятся на связи и вовлечены в рабочий процесс. Кроме того, наличие корпоративной социальные сети − часть организационной культуры, такая же как льготы для сотрудников или мероприятия по тимбилдингу. Так или иначе, корпоративные социальные сети − это несомненно тренд, и наиболее прогрессивные компании ему уже следуют [3].

Отсюда следует что в скором будущем все компании перейдут на корпоративные социальные сети так как они имею ряд преимуществ по сравнению с общедоступными социальными сетям такие как:

1 Разрушение иерархических границ. Не всегда рядовой сотрудник может связаться с кем-то из топ-менеджмента. Социальная сеть компании позволяет работнику напрямую обратиться к руководителю, а начальству - наладить обратную связь с подчиненными.

2 Создание сообщества. Чем крупнее организация, тем меньше работники к ней привязаны. Обычно сотрудник даже не знает, кто работает в соседнем отделе и какие цели преследует компания, в которой он работает. Неформальные связи объединяют работников, делая коллектив более сплоченным.

3 Защита данных от потерь. Иногда для поиска нужной информации приходится пересматривать десятки, если не сотни электронных писем. В социальной сети сотрудник может быстро просмотреть предыдущие переписки в личных чатах и в группах. Кроме того, корпоративные платформы позволяют сохранять контент и предлагают расширенные функции поиска, чтобы работникам не пришлось тратить время на фильтрацию информации.

4 Создание имиджа организации. Чем больше работники уверены в том, что их компания прогрессивная и стабильная, тем качественней они выполняют свои обязанности. Сотрудники приобретают уверенность в завтрашнем дне и понимают, что у них есть перспективы, если знают, какие цели у компании и как быстро она развивается.

5 Контроль поведения сотрудников. Как внутренний корпоративный портал, так и социальная сеть позволяют отслеживать действия и обсуждения работников, чтобы гарантировать соблюдение корпоративной политики организации. С помощью платформы можно наделять сотрудников полномочиями или снимать с них привилегии. Правильно структурированная сеть может стать отличным инструментом, формирующим корпоративную культуру компании [4].

Следовательно, тема создания поддержания и разработка корпоративных социальных сетей является актуальной на текущей момент. Целью данного дипломного проекта является разработка корпоративной социальной сети для унитарного предприятия. Что говорит о том, что тема дипломного проекта является актуальной.

**1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ**

**1.1 Анализ и описания функциональных возможностей программного средства**

В ходе дипломного проектирования необходимо разработать программное средство под операционную систему Android для пересылки информации в режиме реального времени. Программное средство должно представлять собой чат для организации социального взаимодействия между сотрудниками компании.

Основными функциональными требования разрабатываемого программного средства являются:

– регистрация пользователей;

– идентификация пользователей;

– аутентификация и авторизация пользователей;

– обработка данных в удаленной базе данных;

– пересылка сообщений между пользователями в режиме реального времени;

– сохранения историй чатов между пользователей в удаленной базе данных;

– возможность получения имен всех пользователей из удаленной базы данных.

Идентификация – процесс идентификации заключается в проверке существования пользователя. Заходя в личный кабинет или приложение, информационная система проверяет, зарегистрирован ли пользователь. Для этого используется идентификатор, в качестве которого может выступать номер телефона, email, логин или любой другой уникальный признак, закрепленный за конкретным человеком. С помощью идентификатора веб-ресурсы и приложения различают зарегистрированных людей [5].

Система идентификации основывается на одном простом принципе – существование двух одинаковых идентификаторов невозможно. Многие сталкивались с ситуацией, когда при попытке зарегистрироваться, ресурс выдавал ошибку, в которой говорилось, что введенный логин занят.

После того, как информационная система проверила наличие пользователя в базе данных, перед ним возникает другая задача. Необходимо узнать, есть ли у него право заходить в аккаунт. Для решения проблемы была разработана аутентификация.

Аутентификацией – под аутентификацией понимают процесс ввода и последующей проверки пин-кода или пароля. Если он верен, открывается доступ к учетной записи и хранящейся там информации. Аутентификация бывает трех видов:

1 Однофакторная – право на доступ необходимо подтвердить лишь одним способом, например, введя пароль от учетной записи. Такая разновидность является наиболее распространенной.

2 Двухфакторная – чтобы войти в аккаунт одного пароля недостаточно, от пользователя могут потребовать ввести код из уведомления или другую дополнительную информацию. Такой вид применяется в системах, хранящих персональные и прочие важные данные.

3 Трехфакторная – здесь используются более продвинутые методы обеспечения безопасности данных, например, электронные ключи доступа. Как правило они представляют собой отдельные флэш-накопители, которые подключаются к устройству в момент входа в аккаунт. Подобные способы проверки часто встречаются в банковских приложениях.

Авторизация – присвоение конкретной учетной записи определенных привилегий. Если этот процесс прошел успешно, для пользователя открывается доступ к учетной записи. Перед авторизацией стоит еще одна задача – защита системы от изменений, которые могут быть внесены пользователем.

Многие компании ограничивают возможности рабочих компьютеров. Например, работники не могут самостоятельно устанавливать на ПК какой-либо софт. Это может сделать системный администратор, обладающий соответствующими привилегиями. Он входит в систему под своим логином и самостоятельно устанавливает необходимо программное обеспечение.

Эти процессы зависят друг от друга. Один следует за другим: вначале – идентификация, после – аутентификация, и в конце – авторизация. А что произойдет, если один из этапов из этой цепочки убрать?

Без аутентификации проходить идентификацию просто бесполезно. Более того, это может иметь негативные последствия, как для пользователей, так и для владельца ресурса. Если бы не было аутентификации, злоумышленники могли бы без труда получить доступ ко всем личным данным. Для этого им понадобился бы лишь один идентификатор. К примеру, зайти в электронную почту можно было бы без пароля, зная лишь сам адрес. Найти его на просторах всемирной паутины не составляет труда.

Убрав идентификацию, получить доступ к информационной системе станет невозможно. Сайт или приложение просто не поймет, каким образом проводить авторизацию. К слову без авторизации не работает ни один сервис. Все из-за того, что данный процесс тесно связан с функционалом самого ресурса. Если пользователь успешно прошел два предыдущих пункта, но не смог авторизоваться, возникает путаница с правами, которые ему должны быть присвоены. Кроме этого, повышается риск возникновения проблем, связанных с конфиденциальностью.

Предположим, что любой человек может зайти в свой профиль в социальной сети, не проходя при этом авторизации и автоматически получая всевозможные привилегии. В этом случае, каждый сможет просматривать диалоги других пользователей, управлять их профилями, менять настройки и т.д. Именно поэтому везде, где есть личный кабинет, есть и авторизация.

Единственный вариант – авторизация без аутентификации и идентификации. Яркий пример – «Google Документы». Пользователь может разрешить просмотр и изменение документа всем, у кого есть на него ссылка. При ее наличии человек может работать с документом, не проходя аутентификацию и идентификацию.

Для того чтобы пользоваться идентификацией, аутентификацией, авторизацией пользователю необходимо зарегистрироваться. Регистрация пользователя – это создание учетной записи, которая будет храниться на сервере и при помощи, которой пользователь сможет заходить на форум и оставлять сообщения от своего имени [6].

Регистрация пользователей очень важна по нескольким причинам:

1 Защита от спам-ботов. Регистрация позволит если не избавиться полностью от нежелательной рекламной информации, размещаемой специальными программами в блогах и форумах, то значительно сократить её часть.

2 Безопасность посетителей. При регистрации никто не сможет оставить сообщение на форуме от чужого имени. Каждый пользователь будет иметь свой уникальный логин и пароль, которые он будет использовать для того, чтобы оставить сообщение.

3 Регистрация позволит вам вести учёт пользователей форума и привлекать модераторов, которые будут, пользуясь своей учетной записью, удалять или редактировать некорректные сообщения, оставленные другими участниками форума.

Удаленная обработка данных означает передачу, ввод и вывод информации через сеть компьютеров, находящихся на больших расстояниях, как, например, при работе в интернете на тысячах километров. В этом контексте можно выделить несколько форм взаимодействия между компьютерами:

1 Взаимодействие компьютера с удаленным процессом включает обращение с одного компьютера к процессу обработки данных, происходящему на другом компьютере. Это создает логическую связь с процессом и позволяет вести работу с результатами на инициирующем компьютере.

2 Работа с удаленным файлом позволяет открывать, изменять и перемещать файлы, находящиеся на других компьютерах, для последующей локальной обработки.

3 Работа с удаленной базой данных предоставляет доступ к базе данных, хранящейся на другом компьютере, в соответствии с правами доступа пользователя.

4 Взаимодействие компьютеров посредством обмена сообщениями сообщения передаются как отдельным компьютерам, так и группам компьютеров в диалоговом режиме.

5 Электронная почта представляет собой распространенную форму взаимодействия, где каждый абонент имеет почтовый ящик для получения, обработки и пересылки сообщений другим абонентам сети.

В сетевой обработке данных существуют две основные модели:

– децентрализованная обработка основана на решении задач и работе с локальными базами данных на рабочих местах пользователей;

– централизованная обработка включает сервер, который хранит и предоставляет ресурсы, и клиенты, которые имеют удаленный доступ к этим ресурсам.

Серверы могут включать в себя файловые серверы, серверы баз данных, серверы телекоммуникаций, вычислительные серверы, web-серверы и почтовые серверы. Централизованная обработка данных предоставляет доступ к файлам и базам данных многим пользователям, требуя эффективного совместного использования этих ресурсов.

Основные механизмы работы системы мгновенных сообщений включают в себя:

1 Аутентификация и регистрация пользователи должны пройти процесс регистрации и аутентификации для доступа к системе мгновенных сообщений. Обычно это включает создание учетных записей с использованием электронной почты или телефонных номеров.

2 Контактные списки пользователи могут создавать списки контактов, чтобы легко найти и общаться с другими пользователями. Это позволяет быстро и удобно подключаться к нужным собеседникам.

3 Отправка сообщений пользователи могут отправлять текстовые сообщения другим пользователям. Обычно системы мгновенных сообщений предоставляют возможность отправки как одиночных сообщений, так и групповых сообщений, что позволяет общаться одновременно с несколькими людьми.

4 Получение сообщений системы мгновенных сообщений обеспечивают механизмы передачи сообщений от отправителя к получателю. Это может быть осуществлено с помощью централизованного сервера или с использованием протоколов peer-to-peer.

5 Уведомления пользователи получают уведомления о новых сообщениях, чтобы быть в курсе событий и отвечать на них быстрее.

6 Синхронизация сообщений: системы мгновенных сообщений обычно обеспечивают синхронизацию сообщений между устройствами пользователя, что позволяет им открывать чат на одном устройстве и продолжать общение на другом без потери данных.

Системы мгновенных сообщений основаны на различных протоколах, таких как XMPP, MQTT, а также используют различные технологии, включая веб-сокеты, для обеспечения надежной и эффективной передачи сообщений.

Системы мгновенных сообщений являются неотъемлемой частью нашей современной коммуникации и широко используются в различных областях, включая личную переписку, рабочие чаты и даже клиентскую поддержку. Они упрощают обмен информацией и помогают людям оставаться связанными в режиме реального времени.

Для реализации обмена сообщениями в режиме реального времени используется технология WebSocket. WebSocket – это протокол связи поверх TCP, который обеспечивает полдуплексное соединение между браузером и сервером. Это означает, что данные могут передаваться как от браузера к серверу, так и от сервера к браузеру, но не одновременно.

Мессенджеры представляют собой доступное и удобное средство коммуникации, позволяющее пользователям обмениваться сообщениями через интернет. Принцип работы мессенджера через интернет основан на использовании клиент-серверной архитектуры.

Клиент – это приложение мессенджера, установленное на устройстве пользователя. Сервер – это центральная система, которая управляет передачей сообщений между пользователями.

**1.2 Обзор существующих программных средств по теме дипломного проекта**

В ходе дипломного проекта были проанализированы несколько аналогичных приложений типа онлайн мессенджер из онлайн магазина приложений Google Play Store.

Обзор социальной сети Facebook.

Facebook это социальная сеть, функционирующая с февраля 2004 года. Она была обоснована американцем Марком Цукербергом, совместно с группой его соседей по общежитию. Изначально сервис назывался The Facebook и был доступен только представителям Гарварда. Популярность росла, незаметно социальная сеть вышла за пределы одного учебного заведения, а с 2006 года и вовсе стала доступной всем пользователям Интернета [7]. Преимуществами фейсбука являются автоматические правила. Очень полезный функционал как для новичков, так и для профи, который позволяет эффективнее работать с таргетированной рекламой. Больше аналитики. Фейсбук предоставляет расширенную статистику по рекламным кампаниям, а это помогает эффективнее работать с рекламой. Связь с инстаграмом, мы любим, когда с одного кабинета можно управлять продвижением в разных соцсетях. Связь инстаграма и фейсбука позволяет делать рекламные кампании, собирать статистику и анализировать все в одном кабинете. Недостатками же являются локализация. Часть интерфейса фейсбук не переведена на русский язык [8]. Техподдержка фейсбука работает отвечает очень долго и в большинстве случаев не по теме. Сложный дизайн много элементов, мало свободного пространства и кнопки, кнопки, повсюду кнопки, рисунок 1.1.



Рисунок 1.1 – Скриншот пользовательского интерфейса Facebook

Обзор онлайн мессенджера WhatsApp.

WhatsApp – это бесплатное приложение для простого, безопасного и надёжного обмена сообщениями и совершения звонков. Доступно на телефонах по всему миру. Больше 2 миллиардов человек в более чем 180 странах используют WhatsApp, чтобы всегда и везде оставаться на связи с друзьями и близкими [9].

Компания WhatsApp была основана в 2009 году Джаном Кумаром и Брайаном Эктоном. Оба основателя имели опыт работы в Yahoo! и решили создать свою собственную компанию.

Первоначально WhatsApp предлагала возможность обмена сообщениями между пользователями смартфонов через интернет. Уникальность сервиса заключалась в его простоте и легкости использования, а также в отсутствии рекламы.

В начале своей истории компания WhatsApp столкнулась с трудностями, так как они не инвестировали в маркетинг или рекламу. Однако, благодаря положительным отзывам пользователей, WhatsApp начала набирать популярность.

В 2014 году компания была приобретена Facebook за 19 миллиардов долларов, что стало крупнейшей сделкой в истории технологической индустрии на тот момент. После этого приобретения компания начала активно развиваться и привлекать новых пользователей.

Сейчас WhatsApp является одним из самых популярных мессенджеров в мире, с более чем 2 миллиардами активных пользователей. Компания постоянно внедряет новые функции, чтобы улучшить пользовательский опыт и удовлетворить потребности своих пользователей.

WhatsApp обладает такими преимуществами как:

– есть возможность совершать видео и аудио звонки;

– можно отправить media файлы;

– можно отправлять Gif, при чем по умолчанию существует хорошая база гифок;

– сквозное шифрование, все диалоги, чаты сообщения под контролем;

– кастомизация можно поставить себе статус на 24 часа, аватар, изменить фон чат и так далее;

Из минусов можно отметить что:

– отправка файлов максимум 100 МБ;

– временами в фоне не работает или не присылает уведомления;

– на телефонах фирмы Apple расходует излишнее количество батареи;

– возможная небезопасность данных.

Программа позиционируется как заменитель обычным СМС. В большой мере это действительно так. Хотя больше она похожа на чат, и использует для отправки сообщений Интернет. Также кроме текста здесь ты можешь пересылать собеседникам изображения, видео, музыку. Этим она выгодно отличается от традиционных СМС, рисунок 1.2.



Рисунок 1.2 – Скриншот пользовательского интерфейса WhatsApp

Обзор онлайн мессенджера ВКонтакте.

ВКонтакте – это самая большая и популярная социальная сеть в России. Входит в тройку самых посещаемых сайтов Рунета. Сеть является аналогом крупнейшей в мире социальной сети Facebook, которую основал к 2004 году Марк Цукерберг [10].

ВКонтакте создал Павел Дуров в 2006 году. Сначала проект задумывался как площадка для общения сокурсников и выпускников ВУЗов и носило рабочее название – Студент.ру. Но в последствии Павел решил изменить название на более универсальное – "ВКонтакте".

Ресурс создавался с целью возможности общения, поиска людей. Основная идея – оставаться все время на связи в любом месте. Сеть дает возможность создать профиль с фотографией и информацией о себе, пригласить друзей, обменяться с ними сообщениями, оставить сообщения на своей и чужой «стенах», загрузить фотографии, аудио и видеозаписи, создать группы (сообщества по интересам).

ВКонтакте запустился немного позже, чем ее основной конкурент – социальная сеть Одноклассники.ру, но обогнал ее в 2009 году по количеству зарегистрированных пользователей.

В 2008 году ВКонтакте приобрел адрес vk.com для выхода на международный рынок и в 2012 году окончательно перешел на него.

На данный момент сеть считывает более 170 миллионов пользователей. Ежедневная аудитория более 60 миллионов человек.

В сентябре 2014 года, выкупив полный пакет акции, компания Mail.Ru Group стала полноправным владельцем социальной сети ВКонтакте.

ВКонтакте существует возможность размещать таргетированную рекламу. Рекламные объявления показываются в левой части страницы и в ленте новостей. Таргетировать возможно по очень точным параметрам. Оплата происходит за показы или клики.

Приложение социальной сети можно скачать для различных платформ – iOS, Android, Windows Phone.

В социальной сети можно создать свою личную страницу, написать заметку, добавить аудио или видео, общаться с другими участниками Вконтакте, подписываться на обновления групп или пользователей, организовывать встречи или группы. Сначала модераторы сети требовали обязательную регистрацию пользователей под их настоящими именами, но на данный момент на нарушение этого правила смотрят сквозь пальцы.

ВКонтакте предоставляет бесплатный доступ к услугам сети. Заплатить придется только за отправку подарков, размещение рекламы, платные объявления и рейтинг участников.

Оплатить услуги можно посредством Merchant API, разработанной в апреле 2010г. Считалось, что эта платежная система составит достойную конкуренцию Яндекс.Деньги и WM, так как не берет комиссию за покупки в интернет-магазинах, которые часто продавали свои товары участникам ВКонтакте со скидкой от 2% [11].

Долгое время до внедрения рекламы в Рунете ходили слухи, что сеть нужна спецслужбам для сбора личных данных участников ВКонтакте, потому что она имела доступ к большим пластам информации, что не может не оплачиваться. Появление рекламы эти слухи остановило.

ВКонтакте постоянно радует изменениями и новинками в интерфейсе программы, что облегчает навигацию пользователей по сети. ВКонтакте обладает такими преимуществами как:

– вконтакте является самой популярной социальной сетью в России и странах СНГ, имея миллионы активных пользователей это означает, что здесь есть огромная аудитория для ваших контактов, размещения контента и продвижения товаров и услуг [12];

– множество возможностей для общения и развлечений, включая личные сообщения, группы, публикации фотографий и видео, музыку, новости и многое другое;

– удобство использования, интерфейс ВКонтакте прост и понятен даже для новых пользователей;

– платформа предоставляет широкие возможности для хранения и публикации медийного контента, таких как фотографии и видеозаписи, что делает ее привлекательной для творческих индивидуумов и брендов;

– большой ряд инструментов для продвижения бренда или бизнеса, включая создание групп, рекламные возможности и статистику активности аудитории.

Главными минусами ВКонтакте являются:

– ненадежная защита персональной информации, что может повлечь потенциальные угрозы для вашей конфиденциальности и безопасности;

– большое количество спамеров и мошенников так же пользователи могут столкнуться с фишинговыми атаками, рассылкой спама, а также нежелательными рекламными сообщениями;

– отсутствие контроля качества контента что приводи к большому количеству нежелательного контента что может привести к правовым последствиям;

– некорректно прописанный алгоритм что приводит к тому что большинство пользователей жалуются на неактуальность информации.

Пользователи Вконтакте – аудитория с разнообразными интересами. Это привлекает рекламных менеджеров и оптимизаторов. Рекламные кампании в социальной сети результативнее, чем в поисковых системах, так как направлены непосредственно на пользователя без посредников. Участнику ресурса также не запрещается выступать в роли менеджера и рекламировать товар через рекомендации, ссылки или заметки.

Высокий уровень результатов рекламных кампаний помогает достичь вирусный маркетинг, использование которого способно обеспечить регулярный поток потенциальных покупателей. Примером может служить создание групп Вконтакте: чем больше участников в ней состоит, тем эффективнее проходит рекламная кампания.

Программа ВКонтакте выступает в роли ведущей социальной сети в России и странах СНГ, предоставляя миллионам пользователей обширные возможности для общения, развлечений и бизнес-продвижения. С ее постоянными обновлениями интерфейса и привлекательной масштабной аудиторией, ВКонтакте обеспечивает удобство использования и разнообразие инструментов для эффективного взаимодействия. Тем не менее, присутствие проблем в безопасности, таких как недостаточная защита персональной информации и наличие спама, требует от пользователей внимательности и предосторожности при использовании сети, рисунок 1.3.



Рисунок 1.3 – Скриншот пользовательского интерфейса ВКонтакте

Обзор онлайн мессенджера телеграм.

Телеграм – это мессенджер, который позволяет обмениваться сообщениями и многим другим! Это одно из самых популярных приложений для общения в мире.

Он бесплатный и доступен для всех, кто имеет мобильное устройство, подключенное к интернету. Телеграм предлагает множество функций, которые делают его уникальным и привлекательным для пользователей всех возрастных групп.

Основная функция телеграма – это отправка сообщений. Вы можете отправлять сообщения своим друзьям, семье, коллегам – кому угодно, кто также использует это приложение. Кроме того, телеграм позволяет создавать группы и каналы, где вы можете общаться с несколькими людьми сразу. Группы и каналы могут быть открытыми или закрытыми, что дает вам возможность выбирать, с кем вы хотите делиться информацией и кто может присоединиться к вашей беседе.

Помимо обмена текстовыми сообщениями, телеграм также позволяет отправлять различные файлы: фотографии, видео, аудиозаписи, документы и даже геолокацию. Вы можете делиться своими впечатлениями и воспоминаниями с другими людьми, отправлять имм фотографии с праздников или путешествий, смотреть и делиться видео или музыкой. Также телеграм предоставляет возможность создавать и присоединяться к различным сообществам с общими интересами. Вы можете находить и присоединяться к группам пользователей, которые разделяют ваши увлечения, чтобы общаться и делиться информацией по этим темам.

Идея создания телеграма впервые зародилась у Павла Дурова, основателя и бывшего акционера социальной сети «ВКонтакте». По его словам, причиной к запуску мессенджера стало желание создать надежную и безопасную платформу для общения.

В начале своей истории Телеграм предлагал пользователю возможность обмениваться сообщениями, а также использовать групповые чаты и каналы. Однако со временем сервис стал развиваться и добавлять новые функции.

В 2014 году была введена возможность совершать голосовые и видеозвонки. Это позволило пользователям обмениваться не только текстовыми сообщениями, но и голосовыми записями, видео и фотографиями.

В 2015 году Телеграм добавил функцию «Секретные чаты», которая позволяет отправлять самоудаляемые сообщения и устанавливать ограничения на пересылку или копирование контента.

В последующие годы Телеграм развился в более сложную и универсальную платформу. В него были добавлены игры, возможность передачи файлов любого формата, а также функции ботов, позволяющие автоматизировать определенные задачи.

Сегодня телеграм имеет свыше 500 миллионов активных пользователей по всему миру и является одной из наиболее конкурентоспособных платформ для общения. Благодаря своей надежности и безопасности Телеграм привлекает множество пользователей, включая бизнес-сектор и государственные учреждения [13].

Группы в телеграм представляют собой общедоступные чаты, к которым может присоединиться любой пользователь при наличии приглашения или указанных ссылок. Они могут быть открытыми или закрытыми, в зависимости от настроек создателя.

В группе пользователи могут общаться между собой, делиться текстовыми и голосовыми сообщениями, файлами, фотографиями и видео. Также в группе могут быть установлены правила поведения и модерация, чтобы поддерживать порядок и предотвращать нарушения.

Группы могут быть созданы на любую тему, будь то обсуждение фильмов, спортивных событий, обмен опытом в различных областях или ведение бизнеса. Кроме того, в группе могут быть разные уровни доступа для участников, администраторов и модераторов.

Каналы – это односторонние коммуникационные потоки, в которых сообщения отправляются только от создателя к подписчикам. Они часто используются для распространения новостей, информации об акциях, обновлений продуктов и других материалов.

Различные организации, медийные компании, блогеры и общественные деятели часто создают каналы, чтобы донести важные сообщения до широкой аудитории. Пользователи могут подписаться на каналы, чтобы получать уведомления о новых сообщениях и быть в курсе последних событий.

Важно отметить, что каналы не предоставляют возможность напрямую общаться с создателем или другими участниками. Однако пользователи могут комментировать и обсуждать сообщения в канале, если создатель разрешил такую функциональность.

Как и в группах, в каналах можно публиковать текстовые сообщения, фото, видео, аудиозаписи и другие медиафайлы. Также можно установить автоматическую публикацию сообщений из других источников, таких как сайты или другие каналы.

В общем, группы и каналы в телеграм являются мощными инструментами для общения, информационного обмена и организации сообществ. Они позволяют пользователям быть в курсе последних событий, находить единомышленников и получать актуальные и полезные данные.

Во-первых, телеграм обеспечивает конечное шифрование сообщений, что означает, что только отправитель и получатель могут прочитать содержимое сообщения. Дополнительно, в приложении можно настроить функцию «самоуничтожающихся сообщений», при которой отправленные сообщения автоматически удаляются через определенное время после просмотра.

Во-вторых, в телеграме используется двухфакторная аутентификация, которая обеспечивает дополнительный уровень защиты аккаунта. Пользователи могут настроить отправку одноразового кода на дополнительные устройства (например, по смс или через приложение авторизации), чтобы исключить возможность несанкционированного доступа к своей учетной записи.

Кроме того, в телеграме есть функция «каналы», которая позволяет создавать открытые или закрытые сообщества. В открытых каналах любой пользователь может просматривать сообщения, а в закрытых каналах доступ к контенту возможен только по приглашениям. Это позволяет контролировать доступ к информации и ограничивать круг лиц, которые могут видеть определенный контент.

Также важно отметить, что телеграм проводит регулярные проверки на безопасность и пытается оперативно реагировать на выявленные уязвимости. Пользователи могут быть уверены в том, что их данные и переписка находятся в относительно безопасном окружении.

В целом, мессенджер телеграм предлагает разносторонний подход к защите информации и конфиденциальности своих пользователей. Комплексное использование современных технологий шифрования и возможность контроля доступа к содержимому делает его одной из наиболее безопасных платформ для обмена сообщениями, рисунок 1.4.



Рисунок 1.4 –  Скриншот графического интерфейса телеграм

Телеграм – мессенджер, который стал одним из самых популярных в мире благодаря своим многочисленным функциям. Пользователи могут обмениваться текстовыми сообщениями, а также отправлять фотографии, видео, аудиозаписи и документы. Создание групп и каналов позволяет организовать общение с несколькими пользователями одновременно.

Телеграм также обеспечивает возможность создания сообществ с общими интересами и присоединения к уже существующим. Платформа бесплатна и доступна для всех, кто имеет мобильное устройство с интернет-подключением. Основатель, Павел Дуров, выделяет важность безопасности и надежности обмена информацией как ключевые принципы создания приложения.

Таблица 1.1 – Сравнение существующих аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерии | Facebook | WhatsApp | ВКонтакте | Телеграм | Разрабатываемое  программное средство |
| Авторизация | + | + | - | + | + |
| Внутренний  магазин | + | - | + | + | - |
| Реклама | + | - | + | - | - |
| Защита данных | + | + | + | + | + |
| Медиа-контент | + | - | + | - | - |
| Просмотри истории | + | + | + | + | + |

Были рассмотрены существующие аналоги разрабатываемого программного средства. Были выявлены достоинства и недостатки, рассмотрен основной функционал и ключевые особенности каждого из них. После анализа аналогов можно с уверенностью сказать, что для крупных компаний разработка и поддержка своего программного средства является самым безопасным решением.

**1.3** [**Обоснования и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек**](#_Toc132049500)

В качестве языка программирования для реализации программного средства был выбран язык высокого уровня C#.

Основными критериями при выборе языка программирование легли следующие факторы:

– C# современный объектно-ориентированный и строго типизированный язык программирования;

– C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET;

– C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript;

– программы, написанные на языке C#, выполняются в .NET, виртуальной системе выполнения, вызывающей CLR и набор библиотек классов;

– одна общая платформа .NET для разработки всех типов приложений начиная от консольных закачивая кроссплатформенностью.

API на ASP.NET Core.

ASP.NET Core представляет технологию для создания веб-приложений на платформе .NET, развиваемую компанией Microsoft. В качестве языков программирования для разработки приложений на ASP.NET Core используются C# и F# [14].

История ASP.NET фактически началась с выходом первой версии .NET в начале 2002 года и с тех пор ASP.NET и .NET развивались параллельно: выход новой версии .NET знаменовал выход новой версии ASP.NET, поскольку ASP.NET работает поверх .NET. В то же время изначально ASP.NET была нацелена на работу исключительно в Windows на веб-сервере IIS (хотя благодаря проекту Mono приложения на ASP.NET можно было запускать и на Linux).

Однако 2014 год ознаменовал большие перемены, фактически революцию в развитии платформы: компания Microsoft взяла курс на развитии ASP.NET как кроссплатформенной технологии, которая развивается как opensource-проект. Данное развитие платформы в дальнейшем получило название ASP.NET Core, собственно как ее официально именут Microsoft до сих пор. Первый релиз обновленной платформы увидел свет в июне 2016 года. Теперь она стала работать не только на Windows, но и на MacOS и Linux. Она стала более легковесной, модульной, ее стало проще конфигурировать, в общем, она стала больше отвечать требованиям текущего времени.

Текущая версия ASP.NET Core, которая собственно и будет охвачена в текущем руководстве, вышла вместе с релизом .NET 7 в ноябре 2022 года.

Текущую архитектуру платформы ASP.NET Core можно выразить следующим образом, рисунок 1.5.

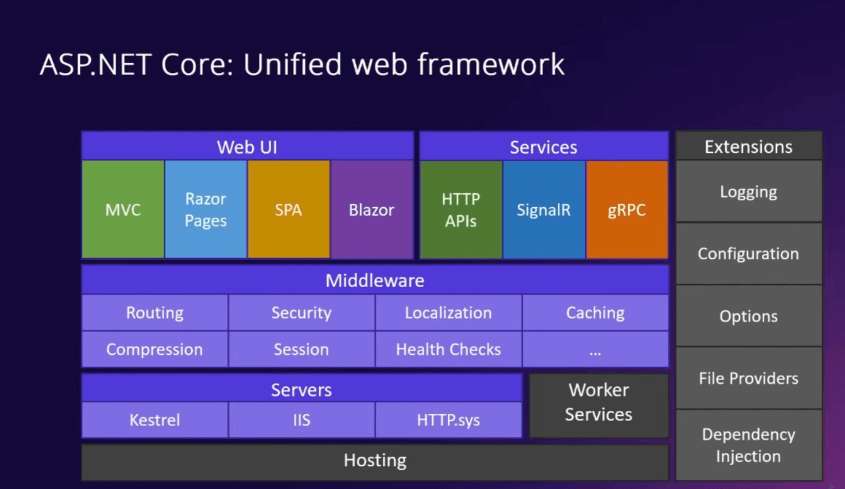


Рисунок 1.5 – Архитектура платформы ASP.NET Core

На самом верхнем уровне располагаются различные модели взаимодействия с пользователем. Это технологии построения пользовательского интерфейса и обработки ввода пользователя, как MVC, Razor Pages, SPA (Single Page Application – одностраничные приложения с использованием Angular, React, Vue) и Balzor. Кроме того, это сервисы в виде встроенных HTTP API, библиотеки SignalR или сервисов GRPC.

Все эти технологии базируются и/или взаимодействуют с чистым ASP.NET Core, который представлен прежде всего различными встроенными компонентами middleware – компонентами, которые применяются для обработки запроса. Кроме того, технологии высшего уровня также взаимодействуют с различными расширениями, которые не являются непосредственной частью ASP.NET Core, как расширения для логгирования, конфигурации и т.д [15].

И на самом нижнем уровне приложение ASP.NET Core работает в рамках некоторого веб-сервера, например, Kestrel, IIS, библиотеки HTTP.sys.

Это вкратце архитектура платформы, теперь рассмотрим моделей разработки приложения ASP.NET Core:

1 Прежде всего это базовый ASP.NET Core, который поддерживает все основные моменты, необходимые для работы соввременного веб-приложения: маршрутизация, конфигурация, логгирования, возможность работы с различными системами баз данных и т.д.

2 ASP.NET Core MVC представляет в общем виде построения приложения вокруг трех основных компонентов – Model (модели), View (представления) и Controller (контроллеры), где модели отвечают за работу с данными, контроллеры представляют логику обработки запросов, а представления определяют визуальную составляющую.

3 Razor Pages представляет модель, при котором за обаботку запроса отвечают специальные сущности - страницы Razor Pages. Каждую отдельную такую сущность можно ассоциировать с отдельной веб-страницей.

4 ASP.NET Core Web API представляет реализацию паттерна REST, при котором для каждого типа http-запроса (GET, POST, PUT, DELETE) предназначен отдельный ресурс. Подобные ресурсы определяются в виде методов контроллера Web API. Данная модель особенно подходит для одностраничных приложений, но не только.

5 Blazor представляет фреймворк, который позволяет создавать интерактивные приложения как на стороне сервера, так и на стороне клиента и позволяет задействовать на уровне браузера низкоуровневый код WebAssembly.

Особенности платформы ASP.NET Core:

– ASP.NET Core работает поверх платформы .NET и, таким образом, позволяет задействовать весь ее функционал;

– в качестве языков разработки применяются языки программирования, поддерживаемые платформой .NET. Официально встроенная поддержка для проектов ASP.NET Core есть у языков C# и F#;

– ASP.NET Core представляет кроссплатформенный фреймворк, приложения на котором могут быть развернуты на всех основных популярных операционных системах: Windows, Mac OS, Linux;

– благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget;

– поддержка работы с большинством распространенных систем баз данных: MS SQL Server, MySQL, Postgres, MongoDB;

– фреймворк построен из набора относительно независимых компонентов;

– в качестве инструментария разработки мы можем использовать такую среду разработки с богатым функционалом, как Visual Studio от компании Microsoft.

Кроссплатформа на .NET MAUI

NET Multi-Platform App UI (.NET MAUI) — это кроссплатформенная платформа для создания собственных мобильных и классических приложений с помощью C# и XAML.

С помощью .NET MAUI можно разрабатывать приложения, которые могут работать на Android, iOS, macOS и Windows из одной общей базы кода, рисунок 1.6.



Рисунок 1.6 – Архитектура .NET MAUI

.NET MAUI является открытым исходным кодом и является эволюцией Xamarin.Forms, расширенной от мобильных до классических сценариев, с элементами управления пользовательским интерфейсом, перестроенными с нуля для повышения производительности и расширяемости. Если вы ранее использовали Xamarin.Forms для создания кроссплатформенных пользовательских интерфейсов, вы заметите множество сходств с .NET MAUI. Однако между ними также есть некоторые различия. С помощью .NET MAUI можно создавать приложения для нескольких платформ в одном проекте и при необходимости добавлять исходный код и ресурсы для конкретной платформы. Одной из основных целей .NET MAUI является возможность реализовать в общей базе кода как можно больше логики и макета пользовательского интерфейса для приложения.

.NET MAUI предназначен для разработчиков, которые хотят:

– написание кроссплатформенных приложений в XAML и C#из одной общей базы кода в Visual Studio;

– совместное использование макета пользовательского интерфейса и разработка на различных платформах;

Приложения .NET MAUI можно записывать на ПК или Mac и компилировать в собственные пакеты приложений:

– приложения Android, созданные с помощью .NET MAUI, компилируются из C# на промежуточный язык (IL), который затем jIT-код компилируется в собственную сборку при запуске приложения;

– приложения iOS, созданные с помощью .NET MAUI, полностью заранее компилируются из C# в собственный код сборки ARM;

– приложения macOS, созданные с помощью .NET MAUI, используют Mac Catalyst, решение от Apple, которое приносит ваше приложение iOS, созданное с помощью UIKit на рабочий стол, и расширяет его с помощью дополнительных API AppKit и платформы по мере необходимости;

– приложения Windows, созданные с помощью .NET MAUI, используют библиотеку Пользовательского интерфейса Windows 3 (WinUI 3) для создания собственных приложений, предназначенных для рабочего стола Windows;

– совместное использование кода, тестов и бизнес-логики на разных платформах.

.NET MAUI предоставляет коллекцию элементов управления, которые можно использовать для отображения данных, запуска действий, указания действий, отображения коллекций, выбора данных и т. д. Помимо коллекции элементов управления, .NET MAUI также предоставляет следующее:

– продуманный механизм визуализации для проектирования страниц;

– несколько типов страницы для создания полнофункциональных типов навигации, таких как панели;

– поддержка привязки данных для создания более элегантного и удобного в обслуживании кода;

– возможность настраивать обработчики для улучшения способа представления элементов пользовательского интерфейса;

– кроссплатформенные API для доступа к собственным функциям устройств.

– кроссплатформенная функциональность графики, которая предоставляет холст рисования, который поддерживает рисование и рисование фигур и изображений, операций создания и преобразования графических объектов;

– единая система проектов, использующая многоцелевой объект для целевой платформы Android, iOS, macOS и Windows;

– горячая перезагрузка .NET, чтобы можно было изменить xaml и управляемый исходный код во время работы приложения, а затем наблюдать за результатом изменений без перестроения приложения.

.NET MAUI предоставляет кроссплатформенные API для собственных функций устройств. Примеры функциональных возможностей, предоставляемых .NET MAUI для доступа к функциям устройства, включают:

– доступ к датчикам, таким как акселерометр, компас и гиро область на устройствах;

– возможность проверка состояния сетевого подключения устройства и обнаруживать изменения;

– копирования и вставка текста в системный буфер для обмена между приложениями;

– выбор одного или нескольких файлов с устройства;

– безопасное хранения данных в виде пар "ключ-значение";

– использования встроенной подсистемы преобразования текста в речь для чтения текста с устройства;

– инициируйте потоки проверки подлинности на основе браузера, которые прослушивают обратный вызов на определенный зарегистрированный URL-адрес приложения.

Библиотека для реализации чата в режиме реального времени SignalR.

ASP.NET SignalR – это библиотека для разработчиков ASP.NET, которая упрощает процесс добавления веб-функций в режиме реального времени в приложения. Веб-функции в режиме реального времени — это возможность мгновенно отправлять содержимое кода сервера на подключенные клиенты по мере его доступности, а не ждать, пока клиент запросит новые данные.

SignalR можно использовать для добавления любых веб-функций в режиме реального времени в приложение ASP.NET. Хотя чат часто используется в качестве примера, вы можете сделать гораздо больше. Каждый раз, когда пользователь обновляет веб-страницу, чтобы увидеть новые данные, или, когда страница реализует длинный опрос для получения новых данных, он является кандидатом для использования SignalR. Примеры включают панели мониторинга и приложения для мониторинга, приложения для совместной работы, обновления хода выполнения заданий и формы в режиме реального времени.

SignalR также включает совершенно новые типы веб-приложений, требующих высокочастотных обновлений с сервера например, игры в режиме реального времени.

SignalR предоставляет простой API для создания удаленных вызовов процедур между серверами (RPC), которые вызывают функции JavaScript в клиентских браузерах (и других клиентских платформах) из кода .NET на стороне сервера. SignalR также включает API для управления подключениями (например, события подключения и отключения) и группирования подключений.

SignalR автоматически управляет подключениями и позволяет транслировать сообщения всем подключенным клиентам одновременно, как в комнате чата. Сообщения можно также отправлять отдельным клиентам. Подключение между клиентом и сервером является постоянным в отличие от классического подключения HTTP, которое устанавливается повторно для каждого сеанса связи.

SignalR поддерживает функцию "отправки сервера", в которой серверный код может вызывать клиентский код в браузере с помощью удаленных вызовов процедур (RPC), а не модель запроса и ответа, распространенную в Интернете сегодня.

Приложения SignalR могут масштабироваться до тысяч клиентов с помощью встроенных и сторонних поставщиков горизонтального масштабирования, рисунок 1.7.



Рисунок 1.7 – Схема вызова процедур между серверами

SignalR использует новый транспорт WebSocket там, где это доступно, и при необходимости возвращается к более старым. Хотя вы, безусловно, можете написать свое приложение с помощью WebSocket напрямую, использование SignalR означает, что большая часть дополнительных функций, которые необходимо реализовать, уже выполнена за вас. Самое главное, это означает, что вы можете накодировать приложение, чтобы воспользоваться преимуществами WebSocket, не беспокоясь о создании отдельного пути кода для старых клиентов. SignalR также защищает вас от необходимости беспокоиться об обновлениях WebSocket, так как SignalR обновляется для поддержки изменений в базовом транспорте, предоставляя приложению согласованный интерфейс в разных версиях WebSocket.

Таким образом все рассмотренные выше технологии являются лучшим выбором для реализации программного средства для пересылки информации в режиме реального времени.

**1.4** [**Постановка задач на дипломное проектирование**](#_Toc132049500)

Реализуемое программное средство направленно на организацию внутреннего коммуникационного взаимодействия между пользователями для реализации данного функционала необходимо определить и решить следующие задачи:

– идентификация пользователя;

– авторизация и аутентификация пользователя;

– регистрация пользователя;

– обработка данных в удаленной базе данных;

– пересылка сообщений между пользователями в режиме реального времени;

– сохранения историй чатов между пользователей в удаленной базе данных;

– возможность получения имен всех пользователей из удаленной базы данных;

– описание и реализация алгоритма по пересылки информации между пользователями;

– тестирования программного средства на наличие ошибок и багов;

– анализ количественных характеристик программного продукта;

– анализ количественных показателей программного средства;

– проектирование руководства пользователя по эксплуатации программного средства;

– определение рекомендованных системных требований программного средства;

– экономически обосновать целесообразность разработки программного средства.

Для решения поставленных задач будет спроектировано социальное кроссплатформенное программное средство для социального взаимодействия с возможностью пересылки информации в режиме реального времени на базе .NET MAUI. В качестве операционных систем, для который будет разработано программное средство были выбраны операционная система Android версии 11, операционная система Windows, операционная система IOS.

Исследуя вопрос о разнице и функционалу социальных сетей большинство задают вопрос что общего между разными социальными сетями. Все социальные сети имеют такие общие характеристик как:

– чат между пользователями;

– простой и понятный дизайн;

– пересылка информации в режиме реального времени;

– идентификация пользователя.

**2 ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**2.1 Проектирование архитектуры и описание состояний программного средства**

Разрабатываемое программное средство представляет собой два различных сервиса связанных между собой. Каждый сервис состоит из отдельных компонентов и их взаимодействия между собой. Каждый компонент отвечает за определенный функционал, рисунок 2.1.

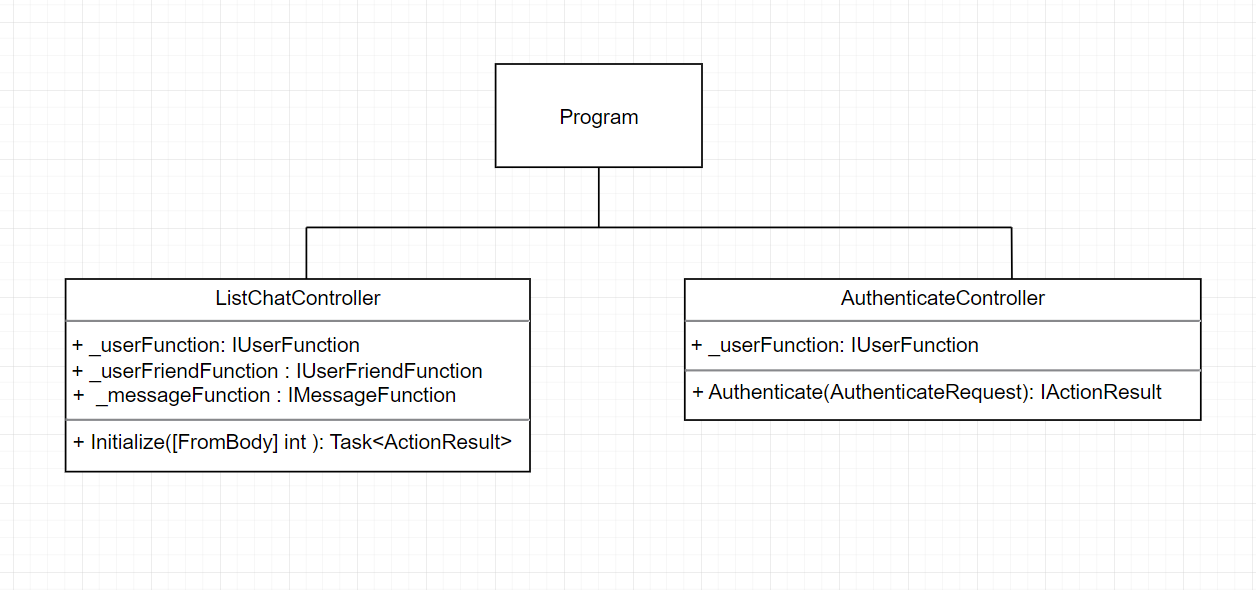


Рисунок 2.1 – Фрагмент UML диаграммы классов отвечающий за работу аутентификации и инициализацию

Класс ListChatController отвечает за вывод на экран информации о всех чатах пользователя в соответствие с информацией, полученной из базы данных. Класс реализует интерфейсы: IUserFunction, IUserFriendFunction и IMessafeFunction которые реализуют методы для получения информации с базы данных. Эти методы реализованы в классах UserFunction, UserFriendFunction и MessafeFunction. Программная реализация данного класса выглядит следующим образом.

﻿using OnlineChatApp.Api.Functions.Message;

using OnlineChatApp.Api.Functions.UserFrined;

namespace OnlineChatApp.Api.Controllers.ListChat

{

[ApiController]

[Route("[controller]")]

public class ListChatController : Controller

{

IUserFunction \_userFunction;

IUserFriendFunction \_userFriendFunction;

IMessageFunction \_messageFunction;

public ListChatController (IUserFunction userFunction, IUserFriendFunction userFriendFunction, IMessageFunction messageFunction)

{

\_userFunction = userFunction;

\_userFriendFunction = userFriendFunction;

\_messageFunction = messageFunction;

}

[HttpPost("Initialize")]

public async Task<ActionResult> Initialize([FromBody] int userId)

{

var response = new ListChatInitializeResponse

{

User = \_userFunction.GetUserById(userId),

UserFriends = await \_userFriendFunction.GetListUserFriend(userId),

LastestMessages = await \_messageFunction.GetLastestMessage(userId),

};

return Ok(response);

}

}

}

Рассмотрим первый метод, реализованный при помощи интерфейса IUserFunction который возвращает нам пользователя по его уникальному идентификатору и присваивает значения все его остальным полям согласно полученным данным из базы данных. Программная реализация данного метода выглядит следующим образом.

public User GetUserById(int id)

{

var entity = \_chatAppContext.TblUsers

.Where(x => x.Id == id)

.FirstOrDefault();

if (entity == null) return new User();

var awayDuration = entity.IsOnline ? "" : Utilities.CalcAwayDuration(entity.LastLogonTime);

return new User

{

UserName = entity.UserName,

Id = entity.Id,

AvatarSourceName = entity.AvatarSourceName,

IsAway = awayDuration != "" ? true : false,

AwayDuration = awayDuration,

IsOnline = entity.IsOnline,

LastLogonTime = entity.LastLogonTime

};

}

Вторым метод GetListUserFriend был реализован при помощи интерфейса IUserFriendFunction который описал в классе UserFriendFunction он возвращает список всех друзей того пользователя чей уникальный идентификатор был получен как входные параметры. Программная реализация данного класса выглядит следующим образом.

﻿namespace ChatAppWithSignalR.Api.Functions.UserFriend

{

public class UserFriendFunction : IUserFriendFunction

{

ChatAppContext \_chatAppContext;

IUserFunction \_userFunction;

public UserFriendFunction(ChatAppContext chatAppContext, IUserFunction userFunction)

{

\_chatAppContext = chatAppContext;

\_userFunction = userFunction;

}

public async Task<IEnumerable<User.User>> GetListUserFriend(int userId)

{

var entities = await \_chatAppContext.TblUserFriends

.Where(x => x.UserId == userId)

.ToListAsync();

var result = entities.Select(x => \_userFunction.GetUserById(x.FriendId));

if (result ==null) result = new List<User.User>();

return result;

}

}

}

Последним реализованным методом является GetLastestMessage это метод реализован при помощи интерфейса IMessageFunction в классе MessageFunction он возвращает на экран устройства последнее сообщения которые были отправлены другим пользователь. Программная реализация данного метода выглядит следующим образом.

public async Task<IEnumerable<LastestMessage>> GetLastestMessage(int userId)

{

var result = new List<LastestMessage>();

var userFriends = await \_chatAppContext.TblUserFriends

.Where(x => x.UserId == userId).ToListAsync();

foreach (var userFriend in userFriends)

{

var lastMessage = await \_chatAppContext.TblMessages

.Where(x => (x.FromUserId == userId && x.ToUserId == userFriend.FriendId)

|| (x.FromUserId == userFriend.FriendId && x.ToUserId == userId))

.OrderByDescending(x => x.SendDateTime)

.FirstOrDefaultAsync();

if (lastMessage != null)

{

result.Add(new LastestMessage

{

UserId = userId,

Content = lastMessage.Content,

UserFriendInfo = \_userFunction.GetUserById(userFriend.FriendId),

Id = lastMessage.Id,

IsRead = lastMessage.IsRead,

SendDateTime = lastMessage.SendDateTime

});

}

}

return result;

}

Класс AuthenticateController отвечает за аутентификацию пользователей в программном средстве, присвоения пользователю JWT токена, определение клеймов. При несоответствии введенных данных с данными в базе данных аутентификация не будет пройдена и пользователь не получит доступ в базу данны. Программная реализация данного класса выглядит следующим образом.

﻿using Microsoft.AspNetCore.Mvc;

namespace OnlineChatApp.Api.Controllers.Authenticate

{

[ApiController]

[Route("[controller]")]

public class AuthenticateController : Controller

{

private IUserFunction \_userFunction;

public AuthenticateController(IUserFunction userFunction)

{

\_userFunction = userFunction;

}

[HttpPost("Authenticate")]

public IActionResult Authenticate(AuthenticateRequest request)

{

var response = \_userFunction.Authenticate(request.LoginId,request.Password);

if (response == null)

{

return BadRequest(new { StatusMessage = "Invalid username or password!" });

}

return Ok(response);

}

}

}

Последним реализованным методом является Authenticate это метод реализован при помощи интерфейса IUserFunction в классе UserFunction возвращает пользователя с присвоенным ему JWT токеном Программная реализация данного метода выглядит следующим образом.

public User? Authenticate(string loginId, string password)

{

try

{

var entity =\_chatAppContext.TblUsers.SingleOrDefault(x=>x.LoginId == loginId);

if (entity == null) return null;

var isPasswordMatched = VertifyPassword (password, entity.StoredSalt, entity.Password);

if (!isPasswordMatched ) return null;

var token = GenerateJwtToken(entity);

return new User

{

Id = entity.Id,

UserName = entity.UserName,

Token = token

};

}

catch (Exception ex)

{

return null;

}

}

Аналогичным методом реализованы другие классы и интерфейсы, отвечающие за весь функционал программного средства. Основной особенностью разработки диаграммы классов при работке WEB API для MAUI является то что при переходе по конечным точкам каждый запрос проходит через так называемую трубу, которая отвечает за весь жизненный цикл Http запроса. В Asp.net Web Api начиная с версии .NET 6.0 весь жизненный цикл находиться в классе Program.

Основными целями такого архитектурного подхода являются:

– простота в использования;

– масштабируемость;

– кроссплатформенность;

– поддерживаемость.

Все выше описанные цели при разработке программного средства являются достаточно сложными в реализации, которая в свою очередь нацелена на простоту обслуживания и расширения для программных устройств разного типа. Данные принципы разработки являются основополагающими при разработке бизнес-решений. Рассматривая возможные решения в данной области можно выделить следующие инструменты:

– внедрения зависимостей (DI);

– ASP.NET Web API

– модель / представления / модель-представления(MVVM).

Внедрения зависимостей.

Внедрения зависимостей решает такую проблему так как сильная связанность классов типовой пример выглядит так сильно связанности показан на рисунке 2.2.

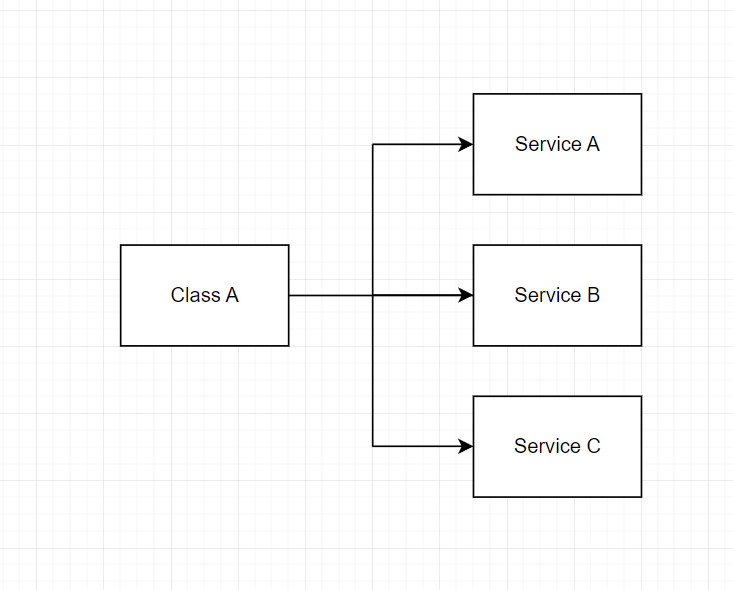


Рисунок 2.2 – Диаграмма компонентов

ClassA имеет строгую зависимость от ServiceA/ServiceB и ServiceC. Такая зависимость говорит о том, что при тестировании ClassA необходимо учитывать реализацию всех сервисов. Внедрение зависимостей (DI –  Dependency injection) представляет механизм, который позволяет сделать взаимодействующие в приложении объекты слабосвязанными. Такие объекты связаны между собой через абстракции, например, через интерфейсы, что делает всю систему более гибкой, более адаптируемой и расширяемой, на рисунке 2.3 изображена реализация прошлого примера с использование внедрения зависимостей.

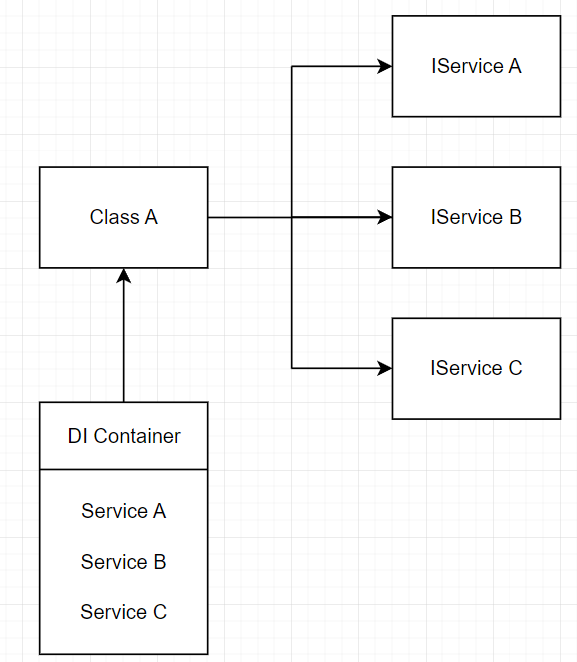


Рисунок 2.3 – Внедрения зависимости

Внедрение зависимостей используется как строитель для генерации ClassA, проверки необходимых зависимостей и их автоматического предоставления. ClassA не заботит то, какой конкретно класс, реализующий требуемый интерфейс, используется, пока таковой имеется в наличии.

В ASP.NET Web API внедрения зависимостей прописывается в классе Program необходимо создать builber класса WebApplication и вызвать метод CreateBuilder после чего необходим добавить зависимости это выглядит следующим образом.

using Microsoft.Extensions.DependencyInjection;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddHostedService<Worker>();

builder.Services.AddSingleton<IUserFunction, UserFunction>();

builder.Services.AddTransient<IMessageFunction, MessageFunction>();

using IHost host = builder.Build();

host.Run();

ASP.NET Web API.

Web API представляет способ построения приложения ASP.NET, который специально заточен для работы в стиле REST. REST-архитектура предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

– GET;

– POST;

– PUT;

– DELET.

По сути Web API представляет собой веб-службу, к которой могут обращаться другие приложения. Причем эти приложения могут представлять любую технологию и платформу –  это могут быть веб-приложения, мобильные или десктопные клиенты. В нашем случае в Web API взаимодействует .NET MAUI кроссплатформенное предложения, которое для реализации функционала которого используется Web API.

Обычно функциональные особенности, реализованные в контролерах. Для этого не обходимо помети что данный контролер выступает как API контроллер. Помечается контроллер атрибутом [ApiController] который позволяет использовать ряд дополнительных возможностей, в частности, в плане привязки модели и ряд других. Также к контроллеру применяется атрибут маршрутизации, который указывает, как контроллер будет сопоставляться с запросами. После того как контроллер был помечен с ним можно работать как с контролером для Web API.

В данном программном средстве его основной функционал заключается в том, чтобы общаться с базой данных и работать с данными полученными из нее. Схема взаимодействие с Web API показана на рисунке 2.4.

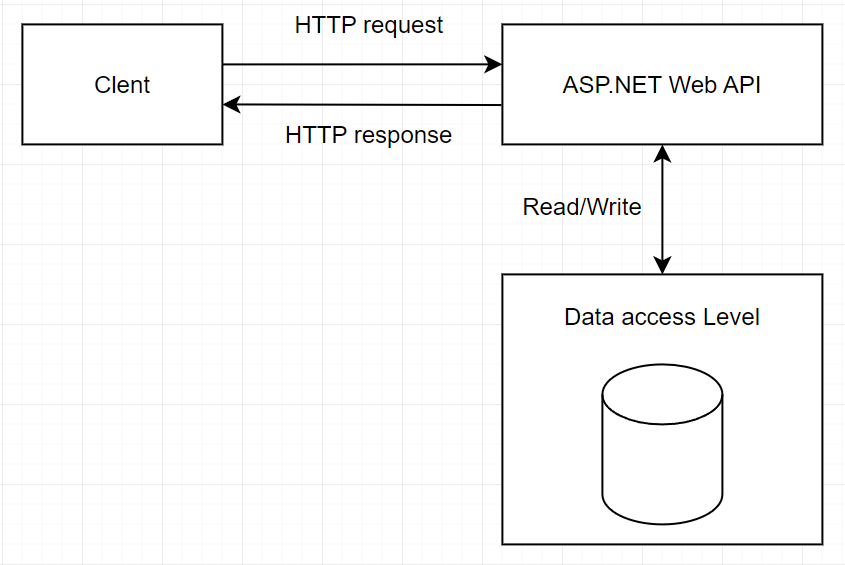


Рисунок 2.4 – Схема взаимодействие с Web API

Как видно из выше предоставленной схемы Web API находиться в центре цыпочки взаимодействия и является основным звеном, которое выполняет весь основной функционал по передаче, получении и работе с данными. Служит для организации работы с удаленной базой данной, а также для связи с клиентом.

Модель / представления / модель-представления (MVVM).

Интерфейс разработчика .NET MAUI обычно включает создание пользовательского интерфейса в XAML, а затем добавление кода позади, которое работает в пользовательском интерфейсе. Сложные проблемы обслуживания могут возникать при изменении и росте размера приложений и область. Эти проблемы включают тесное взаимодействие между элементами управления пользовательским интерфейсом и бизнес-логикой, что повышает затраты на изменение пользовательского интерфейса и трудности модульного тестирования такого кода.

Шаблон MVVM помогает четко отделять бизнес-логику приложения и логику презентации от пользовательского интерфейса. Поддержание четкого разделения между логикой приложения и пользовательским интерфейсом помогает решить многочисленные проблемы разработки и упрощает тестирование, обслуживание и развитие приложения. Он также может значительно улучшить возможности повторного использования кода и позволяет разработчикам и конструкторам пользовательского интерфейса работать более легко при разработке соответствующих частей приложения.

В шаблоне MVVM есть три основных компонента: модель, представление и модель представления. Каждый служит отдельной целью, рисунок 2.5.



Рисунок 2.5 – Схема реализации MVVM паттерна

Помимо понимания обязанностей каждого компонента, важно также понять, как они взаимодействуют. На высоком уровне представление "знает о" модели представления, и модель представления "знает о" модели, но модель не знает о модели представления, и модель представления не знает о представлении. Поэтому модель представления изолирует представление от модели и позволяет модели развиваться независимо от представления.

Преимущества использования шаблона MVVM:

1 Если существующая реализация модели инкапсулирует существующую бизнес-логику, это может быть трудно или рискованно изменить ее. В этом сценарии модель представления выступает в качестве адаптера для классов моделей и не позволяет вносить основные изменения в код модели.

2 Разработчики могут создавать модульные тесты для модели представления и модели без использования представления. Модульные тесты для модели представления могут выполнять точно те же функции, что и в представлении.

3 Пользовательский интерфейс приложения можно изменить без касания модели представления и кода модели, если представление реализовано полностью в XAML или C#. Поэтому новая версия представления должна работать с существующей моделью представления.

4 Верстальщики и разработчики могут работать независимо и параллельно с их компонентами во время разработки. Верстальщики могут сосредоточиться на представлении, в то время как разработчики могут работать над моделью представления и компонентами модели.

5 Ключ к эффективному использованию MVVM заключается в понимании того, как фактор кода приложения в правильные классы и как классы взаимодействуют. В следующих разделах рассматриваются обязанности каждого класса в шаблоне MVVM.

Простой пример реализации MVVM модели показан на рисунке 2.6. В нем используется x:Static расширение разметки для получения текущей даты и времени из статического DateTime.Now свойства в System пространстве имен.

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:sys="clr-namespace:System;assembly=netstandard"

x:Class="XamlSamples.OneShotDateTimePage"

Title="One-Shot DateTime Page">

<VerticalStackLayout BindingContext="{x:Static sys:DateTime.Now}"

Spacing="25" Padding="30,0"

VerticalOptions="Center" HorizontalOptions="Center">

<Label Text="{Binding Year, StringFormat='The year is {0}'}" />

<Label Text="{Binding StringFormat='The month is {0:MMMM}'}" />

<Label Text="{Binding Day, StringFormat='The day is {0}'}" />

<Label Text="{Binding StringFormat='The time is {0:T}'}" />

</VerticalStackLayout>

</ContentPage>

В этом примере извлекаемое DateTime значение задается как значение в объекте StackLayoutBindingContext. При установке BindingContext элемента он наследуется всеми дочерними элементами этого элемента. Это означает, что все дочерние элементы StackLayout одного и того же BindingContext, и они могут содержать привязки к свойствам этого объекта.

**2.2 Формализация предметной области программного средства**

В данной дипломной работе разработано средство для передачи информации в режиме реального времени с минималистичным дизайном и простотой использования. Системный требования являются максимально возможными что говорит о том, что каждый сможет уставить себе программное средство. Прежде чем перейти к использованию чата необходимо будет пройти простую регистрацию, а после авторизацию. В приложении присутствует система оповещений пользователя через шторку уведомлений.

В ходе проектирования функционального процесса было принято решение создать приватный чат между пользователей где пользователь будет выбирать с кем ему вести диалог. Так же просматривать историю сообщений со всеми пользователями.

В ходе проектирования функционального процесса была построена модель разрабатываемого программного средства и придуман интерфейс пользователя.

При разработке приложения применялись следующие функции:

– authenticate – данная функция отвечает за аутентификацию;

– addMessage – функция отправки сообщения;

– getLastestMessage – функция отображения последнего сообщений в окне выбора чатов;

– getMessages – функция получения всех сообщений

– getUserById – функция получения пользователя по его уникальному идентификатору;

– vertifyPassword – функция проверки введенного пароля на верность согласно ранее заданных критерий;

– generateJwtToken – функция создания специального токена для навигации по приложению согласно правам пользователя;

– getListUserFriend – функция для получения списка всех друзей пользователя.

Средства передачи информации в режиме реального времени являются самыми популярными средствами взаимодействия и общения в данный момент.

Данный вид приложения выделяться простым дизайном и схожим функционалом. Обычно у пользователя есть возможность создать свой профиль. Так же иметься возможность заводить друзей обмениваться между ними информацией. Меня картинку профиля пользователя, получать уведомления о то что пользователю кто-то отправил сообщения. Это позволяет не пропустить важную информацию. Приложения подобного типа великое множество, поэтому подобрать подходящий вариант не составит труда.

Изначально приложения по пересылки информации в режиме реального времени разрабатывались студентами для упрощения общения между собой, но в виду их невероятного удобства ими начали пользоваться не только в студенческих целях. Но при этом основная суть данных приложения осталась не изменой.

Одним из востребованных типом приложений является файлообменник. Цель такого приложения обмена файлами. Классический пример – это Skype. Похожие приложения могут предлагать разграничения по типам передаваемых файлов и по их объёму. Но самыми привлекательными считаются те где самый простой и удобный интерфейс в котором отсутствует реклама.

**2.3 Проектирования и реализация способа хранения данных программного средства**

Для реализации хранения данных программного средства пересылки информации в режиме реального времени будет использоваться удаленная база данных **SQL Server.**

SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

SQL Server был создан компанией Microsoft. Первая версия вышла в 1987 году. А текущей версией является версия 2022, которая вышла в ноябре 2022 году.

SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux.

SQL Server характеризуется такими особенностями как:

– производительность;

– надежность и безопасность;

– простота.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных или СУБД. И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ. В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ, рисунок 2.6.

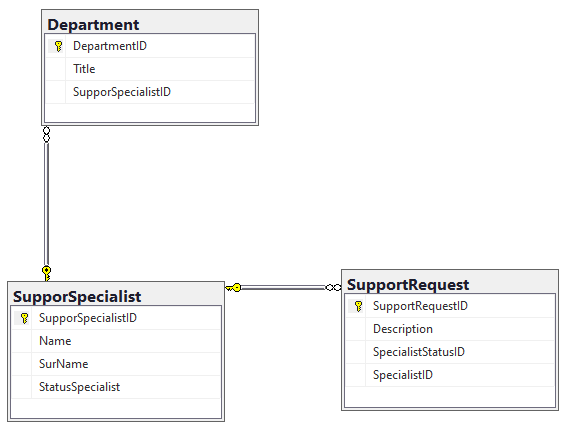


Рисунок 2.6 – Пример реализации связей в MS SQL Server.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения "relation". Для взаимодействия с базой данных применяется язык SQL. Клиент отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Для реализации общения между базой данных и API в ASP.NET используют ORM-технологии. ORM – это технология программирования, которая создает слой между реляционными базами данных и объектно-ориентированными языками программирования без необходимости написания SQL-запросов. Отображение стандартизирует интерфейсы, сокращая количество шаблонов и ускоряя время разработки.

Технология преобразует множество состояний и кодов, создавая структурированную карту, которая помогает разработчикам лучше ориентироваться в структуре базы данных, рисунок 2.7.

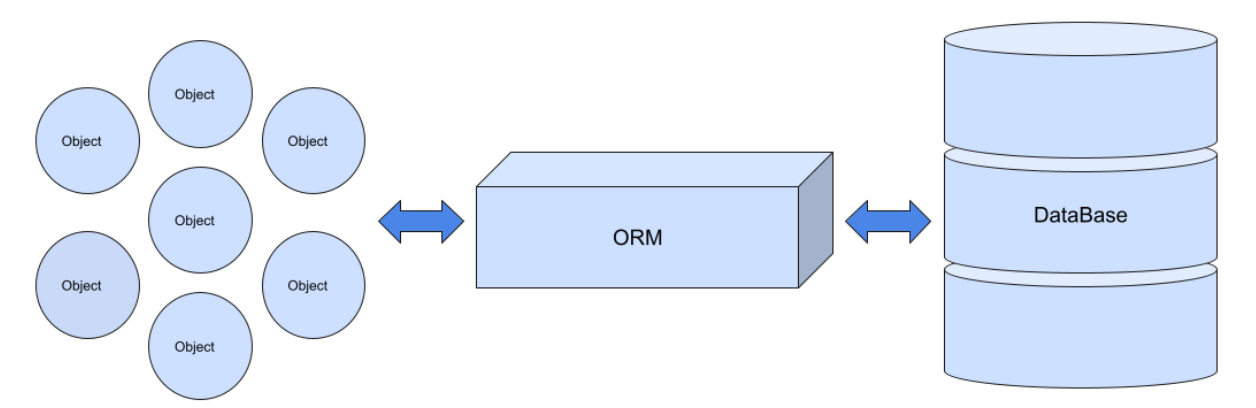


Рисунок 2.7 – Основной прицеп работы ORM.

Отображение объясняет, как объекты связаны с разными таблицами, используя эту информацию для преображения данных между ними. ORM генерирует код SQL для реляционной базы данных в ответ на изменения, которые приложение вносит в объект данных. Сопоставление ORM будет управлять потребностями приложения в данных, избавляя от написания низкоуровневого кода.

ORM выстраивает модели объектно-ориентированной программы с высоким уровнем абстракции. Другими словами, она создает уровень логики без основных деталей кода. Отображение описывает отношения между объектом и данными, не зная, как эти данные структурированы. Далее модель можно использовать для подключения приложения к коду SQL, который необходим для управления операциями с данными. Этот сантехнический тип кода не нужно переписывать, что значительно экономит время разработки.

В данном проекте мы будем использовать такую ORM как Entity Framework. Entity Framework представляет ORM-технологию от компании Microsoft для доступа к данным. Entity Framework Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Как технология доступа к данным Entity Framework работает поверх платформы .NET и поэтому может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

При проектировании и разработки базы данных для данной дипломной работы использовался подход Code First. В подходе Code First фокус идет на приложении. Изначально создаются классы на основе которых в последующим будут созданы сущности в базе данных пример такого класса выглядит следующим образом.

public class TblUser

{

public int Id { get; set; }

public string LoginId { get; set; } = null!;

public string UserName { get; set; } = null!;

public string Password { get; set; } = null!;

public byte[] StoredSalt { get; set; } = null!;

public string AvatarSourceName { get; set;} = null!;

public bool IsOnline { get; set; }

public DateTime LastLogonTime { get; set; }

}

После создания всех моделей необходимо реализовать класс, который будет хранить все модели для реализации этого класса необходимо унаследоватся от класса DbContext после чего реализовать его конструктор следующим образом

public class ChatAppContext : DbContext

{

public ChatAppContext(DbContextOptions<ChatAppContext> options) :base (options)

{ }

public virtual DbSet<TblUser> TblUsers { get; set; } = null!;

public virtual DbSet<TblUserFriend> TblUserFriends { get; set; } = null!;

public virtual DbSet<TblMessage> TblMessages { get; set; } = null!;

}

Следующим этапом необходимо определить ConnectionString в котором будут указаны все параметры необходимые для подключения к базе данных. Обычно ConnectionString записывается в файле с названием appsettings.json после чего необходимо объявить DbContext и ConnectionString в классе Programm. После всего этого необходимо сделать миграцию для того чтобы все модели, которые были реализовали при помощи кода так же были реализованы в базе данных, рисунок 2.8.

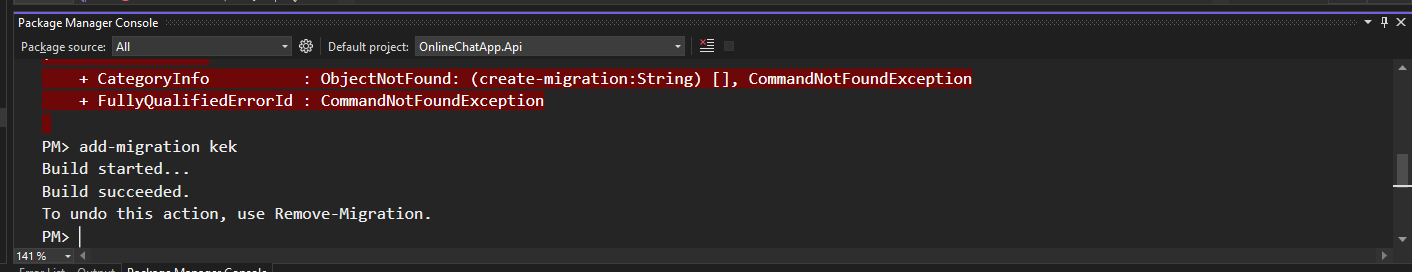


Рисунок 2.8 – Создание миграции при помощи Entity Framework.

После выполнения данной команде создаться директория в которой будут храниться все последующие миграции. В это директории после каждой миграции будет создаваться класс, в котором будут описаны все изменения, которые принесет с собой миграция, рисунок 2.9.

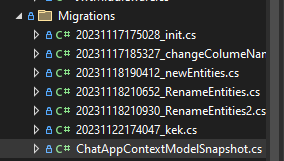


Рисунок 2.9 – Директория, автоматически созданная для миграций.

Последним шагом после проверки что все данные буду созданы верно и нет необходимости производить корректировки в миграцию. Необходимо применить изменения к базе данных, рисунок 2.10.

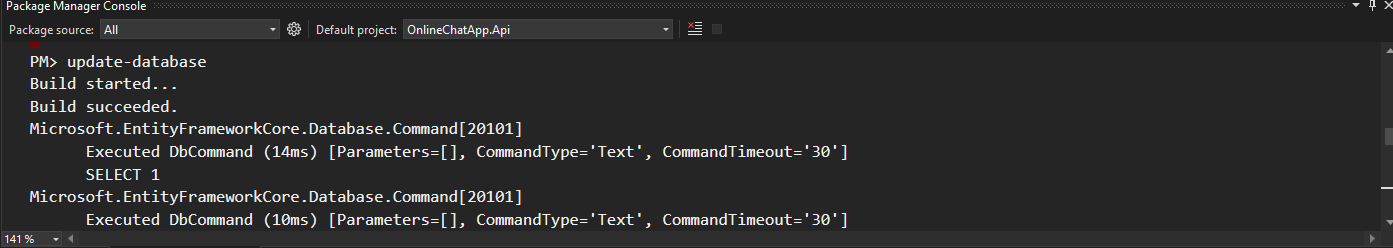


Рисунок 2.10 – Применения миграции к базе данных.

После создания базы данных необходимо проверить действительно ли она была создана. Для этого необходимо зайти в MS SQL Server и проверить там наличие базы данных с заданным название, рисунок 2.11.

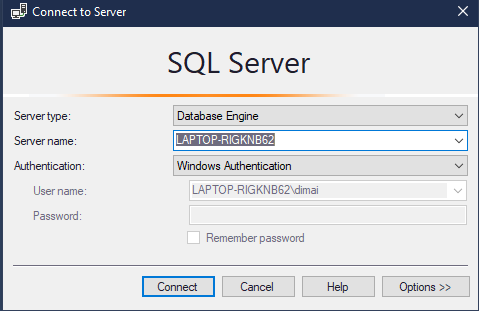


Рисунок 2.11 – Подключение к базе данных через MS SQL Server.

После подключение к серверу и выбора нужной базы данных можно создать диаграмму всех сущностей с их связями что показана на рисунке 2.12.

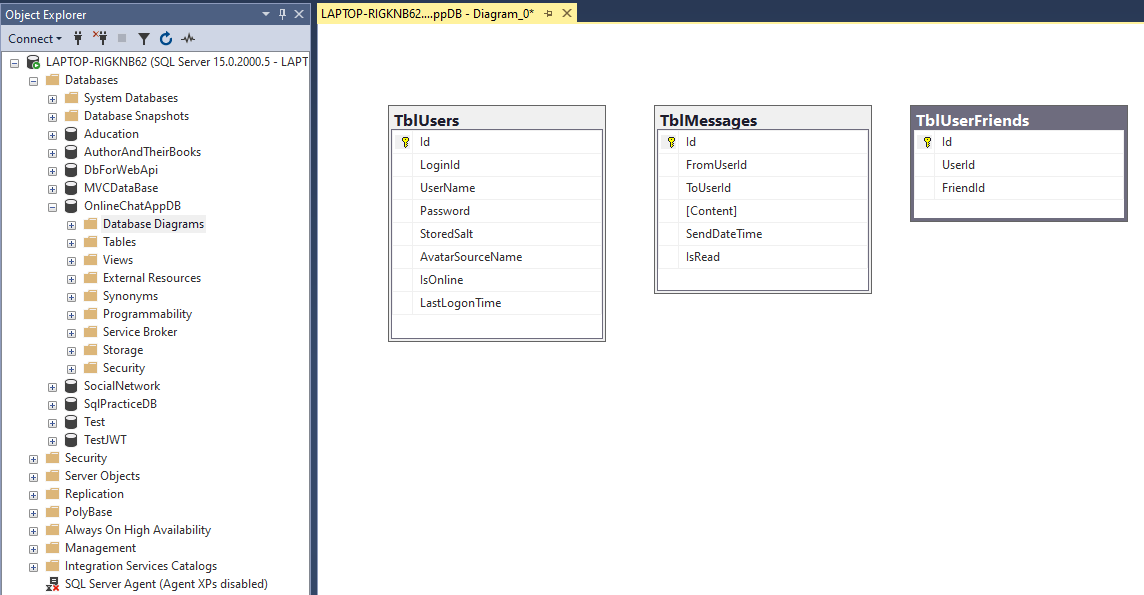


Рисунок 2.12 – Диаграмма сущностей базы данных.

После всех выше описанных действий можно преступать к работе с базой данных. В классе, в котором необходимо использовать базу данных требуется определить переменную, которая будет представлять базу данных. После этого можно работать с этой переменно выполнять операции с данными такие как создания, удаление, обновления и чтения данных из сущностей.

**2.4 Проектирование и разработка графического интерфейса**

Графический интерфейс пользователя – это тип интерфейса, который позволяет пользователям взаимодействовать с компьютером посредством визуальных элементов, таких как значки, кнопки и окна.

При создания пользовательского интерфейса в .NET MAUI сначала необходимо создать файл типа .NET MAUI ContentPage (XAML), рисунок 2.13. Многоплатформенный пользовательский интерфейс приложения .NET MAUI ContentPage отображает одно представление, которое часто представляет собой макет, например Grid или StackLayout, и является наиболее распространенным типом страницы.

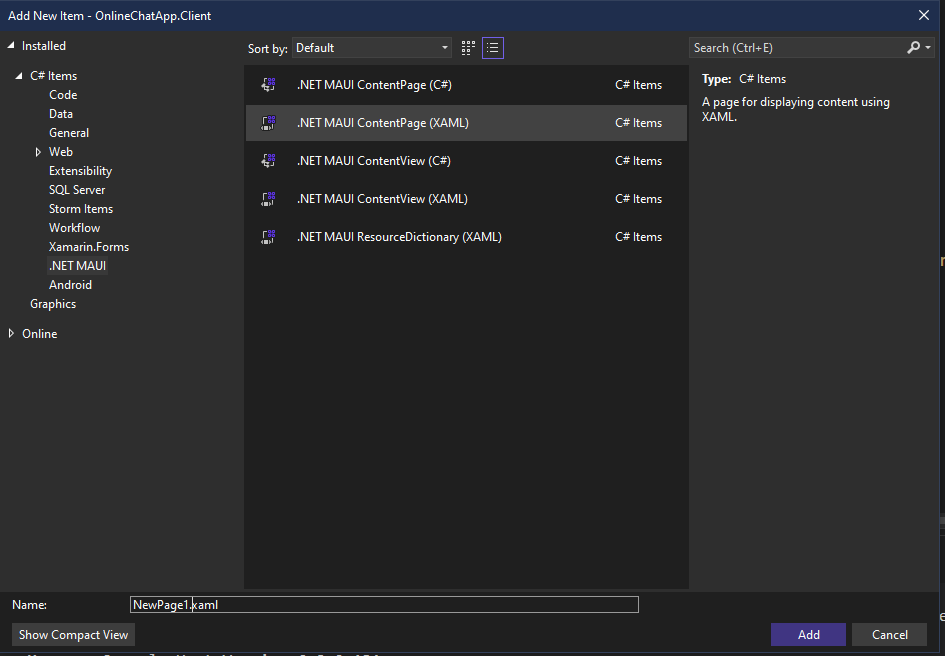


Рисунок 2.13 – Создание ContentPage в .NET MAUI.

Затем Visual Studio создает новую ContentPage производную страницу, XAML код которой будет выглядит следующим образом.

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2023/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class=" OnlineChatApp.Client.Pages.LoginPage"

Title="LoginPage"

BackgroundColor="White">

<StackLayout>

<Label Text="Welcome to .NET MAUI!"

VerticalOptions="Center"

HorizontalOptions="Center" />

<!-- Other views go here -->

</StackLayout>

</ContentPage>

После создания страницы необходимо использовать предать ей красивую оболочку, которая в последующем будет использоваться как страница логина для этого необходимо использовать многоплатформенный пользовательский интерфейс приложения .NET MAUI Grid. Grid – это макет, который упорядочивает дочерние элементы в строки и столбцы, которые могут иметь пропорциональные или абсолютные размеры. По умолчанию Grid содержит одну строку и один столбец. Кроме того, Grid можно использовать в качестве родительского макета, содержащего другие дочерние макеты.

Но Grid не следует путать с таблицами и не предназначен для представления табличных данных. В отличие от HTML-таблиц, Grid он предназначен для размещения содержимого. Для отображения данных использовался VerticalStackLayout и HorizontalStackLayout конечный результа дизайна экрана логина показан на рисунке 2.14.

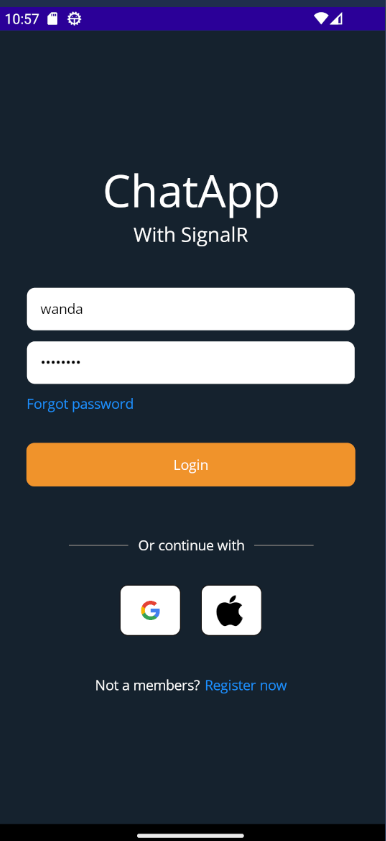


Рисунок 2.14 – Создание пользовательского интерфейса логина

Привязки данных осуществляется при помощи Binding. Привязки данных пользовательского интерфейса приложений .NET (.NET MAUI) позволяют связать свойства двух объектов, чтобы изменение в одном из них приводило к изменению другого. Это очень ценное средство, и хотя привязки данных можно определить полностью в коде, XAML предоставляет сочетания клавиш и удобство.

Привязки данных соединяют свойства двух объектов, называемых источником и целевым объектом. В коде необходимо выполнить два шага:

– свойство BindingContext целевого объекта должно быть задано исходному объекту;

– метод SetBinding (часто используемый в сочетании с Binding классом) должен вызываться в целевом объекте, чтобы привязать свойство этого объекта к свойству исходного объекта.

Целевое свойство должно быть привязываемым свойством, что означает, что целевой объект должен быть производным от BindableObject.

В XAML необходимо также выполнить те же два шага, которые требуются в коде, за исключением того, что Binding расширение разметки занимает место SetBinding вызова и Binding класса. Однако при определении привязок данных в XAML существует несколько способов установки BindingContext целевого объекта. Иногда он устанавливается из файла кода программной части, иногда с помощью StaticResourcex:Static расширения разметки, а иногда и содержимого BindingContext тегов элементов свойства. На странице логина Binding реализован для передачи данных для логина и для запроса с данными в базу данных.

<Frame

Padding="10,0,10,0"

BorderColor="Transparent"

Margin="0,10,0,0">

<Entry

IsPassword="True"

Placeholder="Password"

ReturnType="Go"

TextColor="Black"

Text="{Binding Password}"/>

</Frame>

<Label Margin="0,10,0,0"

Text="Forgot password"

TextColor="#1e90ff"/>

<Button Text="Login"

Background="#f0932b"

TextColor="White"

Margin="0,30,0,0"

Command="{Binding LoginCommand}"/>

<ActivityIndicator

HeightRequest="60"

WidthRequest="60"

IsRunning="{Binding IsProcessing}"/>

Так же было реализовано анимированное кольцо, ожидание которое будет продолжает свою анимацию пока не получит ответ от сервера. Для реализации анимированного кольца использовался ActivityIndicator. В свойствах которого было прописано его поведения при помощи Binding и передачи в него логики поведения, анимированное кольцо показана на рисунке 2.15.

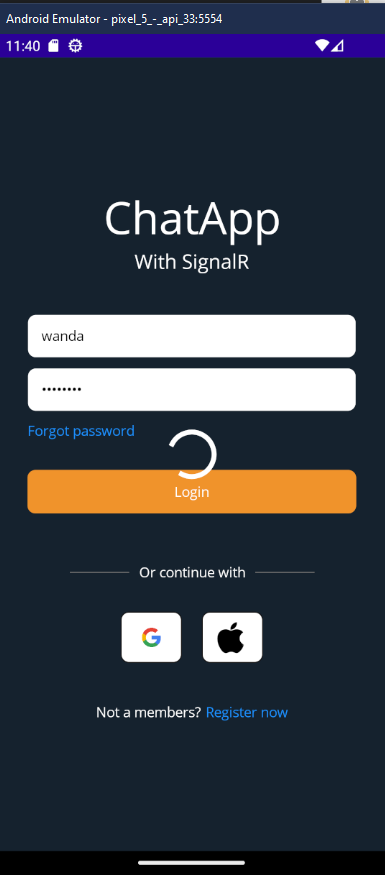


Рисунок 2.15 – Реализация ActivityIndicator на странице логина.

Следующей будет реализована страница чата, в которой будут отображены все приватные чаты с пользователем и все его друзья, с которым пользователь может вести закрытый диалог. Для того чтобы выводить список друзей из базы данных использовался тег CollectionView.

CollectionView в Maui – это элемент управления, который используется для отображения списков данных в пользовательском интерфейсе. Он похож на ListView или RecyclerView из других платформ.

CollectionView обеспечивает более гибкий способ отображения данных, чем традиционные списки, позволяя более тонко настраивать внешний вид и поведение элементов списка. Он предоставляет более высокий уровень контроля над отображением данных и обновлением пользовательского интерфейса.

В Maui CollectionView может использоваться для отображения элементов данных любого типа, будь то текст, изображения, пользовательские элементы управления и так далее. Этот элемент позволяет управлять расположением элементов в списке, их внешним видом и поведением при взаимодействии с пользователем, рисунок 2.16.



Рисунок 2.16 – Реализация страницы чатов.

Аналогичным образов была реализована страница приватного чата. Для межличностного общения между двумя пользователями что является основным функцией разрабатываемого приложения, рисунок 2.17.

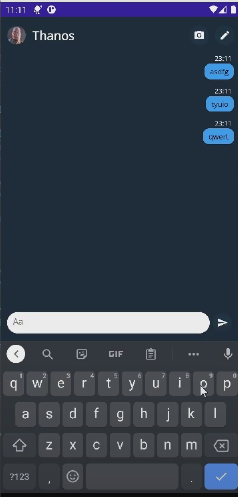


Рисунок 2.17 – Реализация приватного чата.

Таким образом был создан простой, но привлекательный пользовательский интерфейс приложения по пересылки информации в режиме реального времени. Так же были добавлены уведомления для оповещения о новых сообщениях.

**2.5 Описание и реализация используемых в программном средстве алгоритмов**

В данной дипломной работе разрабатывается программное средство пересылки информации в режиме реального времени в котором пользователи могут обмениваться с друг другом сообщениями. Для этого необходимо решить следующие задачи:

– реализовать приватный чат, который будет доступен каждому в котором он будет вести общение только с одним пользователем;

– организовать связь клиента с ASP.NET Web API.

Перейдём к рассмотрению реализации решения поставленных задач.

Алгоритм работы приватного чата в режиме реального времени представлен на рисунке 2.18.

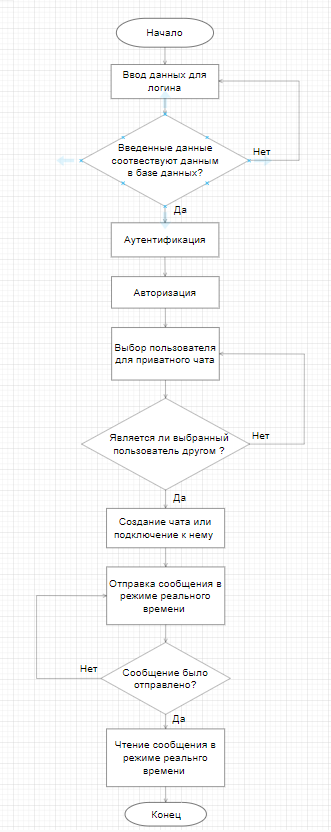


Рисунок 2.18 – Схема алгоритма работы приватного чата.

Для реализации чата в режиме реального времени использовался SignalR. SignalR предоставляет простой API для создания функционала, который позволяет вызывать функции на стороне клиента из серверного кода, написанного с помощью языков платформы .NET. SignalR значительно упрощает работу с коммуникациями реального времени. Библиотека обрабатывает все подключения и автоматически рассылает сообщения всем подключенным клиентам, либо каким-нибудь специфическим клиентам.

Первым делом необходимо создать класс, который будет наследоваться от класса Hub и реализовать там все необходимые методы пример реализации класса выглядит следующим образом.

public class ChatHub:Hub

{

}

Рассмотрим работу класса ChatHub подробно. Первым делом класс наследуется от класса Hub что предоставляет функционал для обмена сообщений между клиентами и сервером. Важно отметить что клиент может обращаться только к публичным методам класс.

Первым делом необходимо создать переменные, которые будут иметь функционал, который реализован при помощи внедрения зависимостей и определили в соответствующих классах и интерфейсах. Так же необходимо определить словарь для сопоставления идентификаторов пользователей с их соответствующими идентификаторами подключения.

UserOperator \_userOperator;

IMessageFunction \_messageFunction;

private static readonly Dictionary<int, string> \_connectionMapping = new Dictionary<int, string>();

Далее необходимо организовать работу метода SendMessage для отправки сообщения от сервера ко всем клиентам. Этот метод вызовут все клиенты, подписанные на событие ReceiveMessage.

public async Task SendMessage(string message)

{

await Clients.All.SendAsync("ReceiveMessage", message);

}

Следующим этапом является основной функционал, организованный в методе SendMessageToUser это метод, который отправляет сообщение от одного пользователя другому в чате. При вызове этого метода указывается идентификатор отправителя, получателя и текст для отправки. Сначала метод ищет получателя согласно входным параметрам, что позволяет определить активные подключения пользователя. Затем он сохраняет сообщение, предположительно, вызывая метод добавления сообщения, и отправляет сообщение.

public async Task SendMessageToUser(int fromUserId, int toUserId, string message)

{

var connectionIds = \_connectionMapping.Where(x=>x.Key == toUserId)

.Select(x=>x.Value).ToList();

await \_messageFunction.AddMessage(fromUserId, toUserId, message);

await Clients.Clients(connectionIds)

.SendAsync("ReceiveMessage", fromUserId, message);

}

Так же необходимо реализовать метод, который будет описывать действия, которые будут происходить при подключении к чату нового пользователя реализован данный функционал в методе OnConnectedAsync.

Этот метод начинается с получения идентификатора пользователя через общий чтобы определить пользователя, который присоединяется к чату. Затем происходит проверка наличия записи о пользователе, в словаре, который был определен как общая переменная. Если записи нет, происходит добавление новой записи, чтобы отслеживать соответствие между пользователем и их подключением к чату. Это делается путем использования идентификатора пользователя в качестве ключа и идентификатора текущего подключения в качестве значения.

public override Task OnConnectedAsync()

{

var userId = \_userOperator.GetRequestUser().Id;

if (!\_connectionMapping.ContainsKey(userId))

\_connectionMapping.Add(userId, Context.ConnectionId);

return base.OnConnectedAsync();

}

Последним этапом является описание того что будет происходить при отключении пользователя из чата. Он просто удаляется из чата при помощи метода remove по его уникальному идентификатору в методе OnDisconnectedAsync. Полный код класса выглядит следующим образом:

public class ChatHub:Hub

{

UserOperator \_userOperator;

IMessageFunction \_messageFunction;

private static readonly Dictionary<int, string> \_connectionMapping

= new Dictionary<int, string>();

public ChatHub(UserOperator userOperator, IMessageFunction messageFunction)

{

\_userOperator = userOperator;

\_messageFunction = messageFunction;

}

public async Task SendMessage(string message)

{

await Clients.All.SendAsync("ReceiveMessage", message);

}

public async Task SendMessageToUser(int fromUserId, int toUserId, string message)

{

var connectionIds = \_connectionMapping.Where(x=>x.Key == toUserId)

.Select(x=>x.Value).ToList();

await \_messageFunction.AddMessage(fromUserId, toUserId, message);

await Clients.Clients(connectionIds)

.SendAsync("ReceiveMessage", fromUserId, message);

}

public override Task OnConnectedAsync()

{

var userId = \_userOperator.GetRequestUser().Id;

if (!\_connectionMapping.ContainsKey(userId))

\_connectionMapping.Add(userId, Context.ConnectionId);

return base.OnConnectedAsync();

}

public override Task OnDisconnectedAsync(Exception? exception)

{

\_connectionMapping.Remove(\_userOperator.GetRequestUser().Id);

return base.OnDisconnectedAsync(exception);

}

}

Таким образом был создан класс, который реализует основные функции двух типов чатов приватного и общественного. Данный способ реализации является простым и максимально эффективным. В результате была решена первая поставленная задача.

На рисунке 2.19 показан алгоритм связи клиента и сервера.

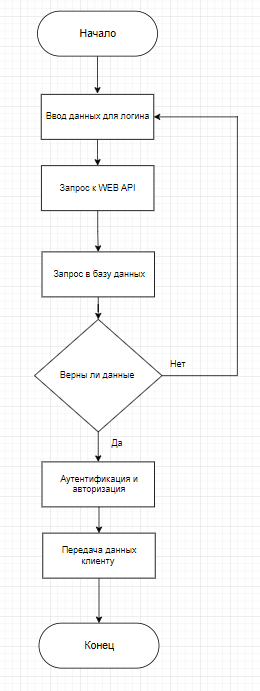


Рисунок 2.19 – Схема алгоритма работы связи клиента и сервера.

Для реализации данного алгоритма был создан класс ServiceProvider в нем будут реализованы методы по аутентификации и по запросу к Web API. Для него так же будет использоваться дополнительные классы такие как DevHttpsConnectionHelper для подключения к локальному серверу.

Первым делом необходимо объявить переменную, в которой будет храниться токен который в дальнейшем будет использоваться для авторизации и аутентификации при помощи технологии JWT. Второй переменной будет вспомогательная переменная кто будет содержать конечный локальный адрес для подключения к локальному серверу.

public string \_accessToken = "";

private DevHttpsConnectionHelper \_devSslHelper;

Далее необходимо указать локальный порт подключения его можно узнать при запуске приложения в адресной строке или в файле с название launchsettings.json. Присвоение локального адреса произведено в конструкторе класса.

public ServiceProvider()

{

\_devSslHelper = new DevHttpsConnectionHelper(sslPort: 7093);

}

Следующим шагом является реализация метода Authenticate который реализует аутентификацию пользователя. Первым делом происходит инициализация HTTP-запроса, который будет отправлен на сервер. В нем создается новы экземпляр HttpRequestMessage метод запроса устанавливается как POST. Затем происходит формулировка URI запроса который состоит из корневого адреса и конечной точки которая представляет собой путь к Web API.

var httpRequestMessage = new HttpRequestMessage();

httpRequestMessage.Method = HttpMethod.Post;

httpRequestMessage.RequestUri = new Uri(\_devSslHelper.DevServerRootUrl + "/Authenticate/Authenticate");

После этого необходимо сериализовать запрос в формат Json для предоставления данных запроса для аутентификации пользователя. Что бы реализовать данный функционал необходимо использовать библиотеку Newtonsoft.Json. Затем создается объект, который представляет содержимое HTTP-запроса. Сериализованные данные устанавливаются как содержимое запроса, используя кодировку UTF-8 и указывая тип контента. Это содержимое запроса присваивается в свойство Content объекта httpRequestMessage, готовя запрос к отправке на сервер для выполнения аутентификации пользователя. Так же прежде чем выполнять действия по сериализации запроса необходимо сделать проверку на то за запрос не является пустым.

string jsonContent = JsonConvert.SerializeObject(request);

var httpContent = new StringContent(jsonContent, encoding: Encoding.UTF8, "application/json"); ;

httpRequestMessage.Content = httpContent;

Заключительным этапом будет отправка HTTP-запроса на сервер и обрабатывает полученный с сервера ответ. Сначала запрос отправляется асинхронно на сервер. После получения ответа его содержимое читается как текст. Полученный текст, содержащий ответ от сервера в формате JSON, затем десериализуется в объект нужного типа. После проверяется статус код ответа, и в случае успеха извлекается токен доступа и сохраняется. Наконец, объект, содержащий данные ответа и статус код, возвращается в качестве результата выполнения метода.

var response = await \_devSslHelper.HttpClient.SendAsync(httpRequestMessage);

var responseContent = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<AuthenticateResponse>(responseContent);

result.StatusCode = (int)response.StatusCode;

if (result.StatusCode == 200)

{

\_accessToken = result.Token;

}

return result;

Последним этапом необходимо обернуть весь код в try/catch блок для предотвращения нежелательных остановок приложения. Финальный вариант метода выглядит следующим образом:

public async Task<AuthenticateResponse> Authenticate(AuthenticateRequest request)

{

var httpRequestMessage = new HttpRequestMessage();

httpRequestMessage.Method = HttpMethod.Post;

httpRequestMessage.RequestUri = new Uri(\_devSslHelper.DevServerRootUrl + "/Authenticate/Authenticate");

if (request != null)

{

string jsonContent = JsonConvert.SerializeObject(request);

var httpContent = new StringContent(jsonContent, encoding: Encoding.UTF8, "application/json"); ;

httpRequestMessage.Content = httpContent;

}

try

{

var response = await \_devSslHelper.HttpClient.SendAsync(httpRequestMessage);

var responseContent = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<AuthenticateResponse>(responseContent);

result.StatusCode = (int)response.StatusCode;

if (result.StatusCode == 200)

{

\_accessToken = result.Token;

}

return result;

}

catch (Exception ex)

{

var result = new AuthenticateResponse

{

StatusCode = 500,

StatusMessage = ex.Message

};

return result;

}

}

Следующим методом в классе ServiceProvider реализован метода CallWebApi который после авторизации и аутентификации возвращает данные с сервера.

Первым делам необходимо сформировать Http-запрос для отправки на сервер методом Post c установкой в header токена для аутентификации.

var httpRequestMessage = new HttpRequestMessage();

httpRequestMessage.Method = HttpMethod.Post;

httpRequestMessage.RequestUri = new Uri(\_devSslHelper.DevServerRootUrl + apiUrl);

httpRequestMessage.Headers.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_accessToken);

Следующим этапом происходит проверка того что запрос на сервер не является пустым. Если он не пустой он собирает запрос на сервер для получения списка друзей.

if (request != null)

{

string jsonContent = JsonConvert.SerializeObject(request);

var httpContent = new StringContent(jsonContent, encoding: Encoding.UTF8, "application/json"); ;

httpRequestMessage.Content = httpContent;

}

Далее необходимо получи ответ от сервера и десериализовать его из формата JSON в объект типа TResponse. Код статуса ответа сервера присваивается свойству StatusCode созданного объекта, а затем этот объект возвращается в качестве результата выполнения метода.

var response = await \_devSslHelper.HttpClient.SendAsync(httpRequestMessage);

var responseContent = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<TResponse>(responseContent);

result.StatusCode = (int)response.StatusCode;

return result;

Последним этапом необходимо обернуть весь код в try/catch блок для предотвращения нежелательных остановок приложения. Финальный вариант метода выглядит следующим образом:

public async Task<TResponse> CallWebApi<TRequest, TResponse>(

string apiUrl, HttpMethod httpMethod, TRequest request) where TResponse : BaseResponse

{

var httpRequestMessage = new HttpRequestMessage();

httpRequestMessage.Method = HttpMethod.Post;

httpRequestMessage.RequestUri = new Uri(\_devSslHelper.DevServerRootUrl + apiUrl);

httpRequestMessage.Headers.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_accessToken);

if (request != null)

{

string jsonContent = JsonConvert.SerializeObject(request);

var httpContent = new StringContent(jsonContent, encoding: Encoding.UTF8, "application/json"); ;

httpRequestMessage.Content = httpContent;

}

try

{

var response = await \_devSslHelper.HttpClient.SendAsync(httpRequestMessage);

var responseContent = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<TResponse>(responseContent);

result.StatusCode = (int)response.StatusCode;

return result;

}

catch (Exception ex)

{

var result = Activator.CreateInstance<TResponse>();

result.StatusCode = 500;

result.StatusMessage = ex.Message;

return result;

}

}

Таким образом были реализованы методы для взаимодействия клиента с сервером по средствам создание специального класса, который отвечает за отправку и обработку запросов. Преимуществом такого подхода является простота реализации и высокая надежность.

**2.6 Тестирования программного средства**

Когда программисты создают новое приложение или вносят изменения в существующее, они могут допускать ошибки. Тестирование помогает выявить эти проблемы и убедиться, что приложение работает так, как задумано.

Тестирование – это проверка программного обеспечения, которая показывает, соответствует ли оно ожиданиям разработчиков и правильно ли работает.

Тестирование проводят тестировщики они отвечают за обеспечение качества, контролируют его и проверяют, что продукт соответствует всем заданным требованиям.

Процесс работы над продуктом включает в себя множество этапов: от проработки идеи и расчета эффективности до самой разработки, и выпуска. И в этом процессе участвует множество людей: аналитики, руководители проекта, разработчики, дизайнеры.

Представьте, что все эти люди объединяются, чтобы создать какой-то продукт. Они разрабатывают его, выкатывают на прод. А позже пользователи вдруг выясняют, что где-то в продукте есть баги. В результате команде придется заново его прорабатывать, что стоит немалых денег и времени, да и репутация продукта на рынке будет испорчена.

Устранить ошибки можно заранее, доверив эту работу тестировщикам. Они должны участвовать во всем цикле создания программного обеспечения: от появления требований к проекту до момента сопровождения самого ПО.

Есть несколько этапов тестирования:

1 Проработка требований к продукту – это этап на котром тестировщики внимательно изучают требования продукта – это могут быть документы, спецификации, описание того, как пользователь взаимодействует с продуктом (по-другому это называют пользовательскими сценариями). Четкое понимание требований помогает определить области, которые нужно протестировать.

2 Анализ требований – этот этап позволяет выяснить, какие возможные риски или сложности могут возникнуть при тестировании. Также на этом этапе можно выявить возможные несоответствия или недостаточно ясные требования, которые требуют уточнения у разработчиков или заказчика.

3 Разработка стратегии и плана тестирования.Когда все требования к продукту понятны, остается разработать план тестирования. В него входит:

– выбор методов тестирования ручное, автоматизированное, тестирование на реальных устройствах и другие;

– анализ потенциальных рисков, которые могут повлиять на качество и успешность тестирования, и планирование мер по их минимизации;

– планирование ресурсов – кто будет тестировать продукт, каким оборудованием и инструментами можно при этом пользоваться и сколько времени займет тестирование, к какому сроку оно должно быть закончено.

4 Создание тестовой документации на этом этапе на основе требований и анализа тестировщики создают тестовые случаи, тест-планы, отчетность и другую документацию, которая будет использоваться во время тестирования. Тестовая документация определяет, какие тесты будут проведены, как будут собраны результаты и как будет оценено качество ПО.

5 Тестирование. После того как команда утверждает стратегию тестирования и тестовую документацию, проводится тестирование. Тестирование программного обеспечения – это длительный и обширный процесс. По ходу составляются отчеты о выявленных недостатках, проводится набор тестовых сценариев, создается тестовая среда и выполняется тестирование согласно заранее задокументированным видам тестов, описанным в тестовой документации. Важно понимать, что найти все ошибки в продукте невозможно. Главная цель заключается не в создании идеального продукта без ошибок, а в обнаружении максимального числа дефектов, которые могут потенциально повлиять на работу системы.

6 Эксплуатация и поддержка. После того как разработчики устраняют дефекты и выпускают продукт, тестировщик переходит к тестированию продукта в рабочей среде. Важно отметить, что на этом этапе не только происходит релиз продукта, но и начинается пост-релизовая поддержка.

Невозможно предусмотреть все особенности использования и окружение, в котором будет работать продукт. Поэтому на данном этапе акцент делается на обратной связи пользователей. Теперь они становятся главными тестировщиками, а продукт становится частью их повседневной жизни. Устранение дефектов и поиск ошибок проводится быстро, но тщательно.

Есть множество видов тестирования рассмотрим самые важные.

Ручное тестирование – это проверка программного обеспечения вручную, без использования автоматизированных инструментов. Тестировщик взаимодействует с программой как обычный пользователь.

В ходе ручного тестирования тестировщик выполняет различные сценарии использования и тестовые сценарии, вводит данные, наблюдает за результатами и проверяет, нет ли ошибок или неожиданного поведения. Если обнаруживаются проблемы, тестировщик документирует их, чтобы разработчики могли исправить ошибки.

Ручное тестирование позволяет проверить различные аспекты программы: удобство использования, внешний интерфейс, а также воспроизводить нестандартные ситуации, которые может быть сложно автоматизировать.

Функциональное тестирование проверяет соответствие программы или системы заранее определенным функциональным требованиям и ожиданиям. Основная цель функционального тестирования – убедиться, что программа выполняет свои функции и операции согласно спецификациям, а также работает правильно и без сбоев.

Дымовое тестирование – это быстрая проверка программного обеспечения, которую выполняют после внесения значительных изменений или обновлений в код. Этот вид тестирования напоминает «пробный пуск» программы, чтобы убедиться, что основные функции работают без критических ошибок.

В ходе разработки дипломной работы были учтены все возможные факторы для того что оставить минимально количество багов. Однако тестирование было проведено. Для тестирования был выбран смартфон «SAMSUNG S21 FE» с версией операционной системы Android 13.0.0.

Для определения корректно ли работает программное средство необходимо произвести Альфа тестирования по следующим критериям:

– подготовка альфа-версии приложения включает в себя компиляцию приложения и подготовку к развертыванию на тестовом устройстве для первоначального тестирования;

– провести первичное тестирование среди ограниченной группы пользователей для выявления базовых проблем функциональности и стабильности приложения;

– проверка основного функционала таких как пересылка информации в режиме реального времени, авторизация и т.д;

– тестирование UX/UI на оценку пользовательского интерфейса и опыта, а также проверку на интуитивность, понятность и удобство навигации;

– тестирование совместимости и производительности включает проверку функционирования приложения на различных устройствах и версиях Android, оценку производительности, времени отклика, оптимизацию ресурсов и обработку ошибок;

– проверка уведомлений на корректность и своевременность отображения;

– проверка на уязвимости безопасности, включая защиту данных пользователей, шифрование и аутентификацию.

В результате проведения Альфа-тестирования не было выявлено критических ошибок выявлено не было и были достигнуты следующие результаты:

– при компиляции и развертывание приложения на тестовом устройстве ошибок выявлено не было;

– при первичном тестирование не было выявлено критических ошибок;

– функции пересылки информации в режиме реального времени работала без перебоев;

– пользовательский интерфейс имеет понятный функционал и не вызывает никаких нареканий;

– на различных версиях Android приложения запускалась стабильно;

– на различных устройствах приложения запускалась стабильно;

– всплывающие окна отображали корректную и своевременную информацию.

Все функции приложения на двух устройствах работают исправно без нареканий без лишних нагрузок на систему, зависаний и лишних процессов.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

[1] Леви Стивен https://alpinabook.ru/catalog/book-sotsialnaya-set-izmenivshaya-mir/

[2] https://premiummanagement.com/blog/socialnye-seti-na-rabochem-meste

[3] https://inside-pr.ru/archives/5024

[4] https://www.daoffice.ru/posts/funkczii-zadachi-i-preimushhestva-vnutrennih-soczialnyh-setej#:~:text=%D0%92%20%D0%BE%D1%82%D0%BB%D0%B8%D1%87%D0%B8%D0%B5%20%D0%BE%D1%82%20%D0%BE%D0%B1%D1%8B%D1%87%D0%BD%D0%BE%D0%B9%20%D1%81%D0%BE%D1%86%D1%81%D0%B5%D1%82%D0%B8,%D0%BE%D0%B1%D1%89%D0%B0%D1%82%D1%8C%D1%81%D1%8F%2C%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D1%83%D1%8F%20%D0%BE%D1%84%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D1%80%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B9%20%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82.

[5] https://optimalgroup.ru/blog/identifikaciya-autentifikaciya-avtorizaciya/

[6] https://dic.academic.ru/dic.nsf/ruwiki/1857816

[7] https://perfluence.net/blog/article/zachem-nuzhen-Facebook

[8] https://hiconversion.ru/target/nedostatki-i-preimushchestva-facebook/

[9] https://w-hatsapp.ru/preimushestva-plusy-minusy/

[10] https://www.glossary-internet.ru/terms/%C2/vkontakte/

[11] https://wiki.rookee.ru/vkontakte/

[12] https://uchet-jkh.ru/i/plyusy-i-minusy-plavaniya-vo-vkontakte-preimushhestva-i-riski/

[13] https://uchet-jkh.ru/i/cto-znacit-messendzer-telegramm/

[14] https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/

[15] https://metanit.com/sharp/aspnet6/12.1.php

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программы**