

СОДЕРЖАНИЕ

Введение.....	7
1 Анализ данных и постановка задач на дипломную работу	9
1.1 Анализ исходных данных к дипломной работе	9
1.2 Обзор существующих программных средств по теме дипломной работы.....	13
1.3 Обоснования и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек.....	21
1.4 Постановка задача на дипломную работу	28
2 Проектирование, разработка и тестирование программного средства	29
2.1 Проектирование архитектуры и описание состояний программного средства	29
2.2 Формализация предметной области программного средства	38
2.3 Проектирование и реализация способа хранения данных программного средства	39
2.4 Проектирование и разработка графического интерфейса	44
2.5 Описание и реализация используемых в программном средстве алгоритмов	50
2.6 Тестирования программного средства	57
3 Оценка количественных показателей функционирования программного средства.....	61
3.1 Оценка временных показателей программного средства	61
3.2 Оценка ресурсных показателей программного средства.....	63
3.3 Оценка показателей надёжности программного средства.....	66
4 Эксплуатация программного средства.....	69
4.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию	69
4.2 Рукноводство по эксплуатации программным средством.....	72
5 Техничко-экономическое обоснование разработки программного средства под операционную систему Android для организации онлайн чата с возможностью пересылки информации в режиме реального времени.....	76
5.1 Характеристика программного средства, разрабатываемого для реализации на рынке	76
5.2 Расчет инвестиции в разработку программного средства для его реализации на рынке	77
5.3 Оценка экномического эффекта от реализации программного средства на рынке.....	80

5.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке.....	81
Заключение	84
Список использованных источников	85
Приложение А (обязательное) тчёт о проверке на заимствования в системе «Антиплагиат»	87
Приложение Б (обязательное) Листинги программного кода	88
Приложение В (обязательное) Графический материал, поясняющий разработанное программное средство.....	100
Приложение Г (обязательное) Ведомость дипломной работы.....	106

ВВЕДЕНИЕ

В настоящее время приложения для социального взаимодействия активно развиваются. 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 общения. 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 сетей 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 социальной 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 имеют ряд преимуществ по сравнению с общедоступными социальными сетями 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 текущий момент. Целью данной дипломной работы является разработка программного средства под операционную систему android для пересылки информации в режиме реального времени.

Дипломная работа выполнена самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 87.78 %. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот приведен в приложении.

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ НА ДИПЛОМНУЮ РАБОТУ

1.1 Анализ исходных данных к дипломной работе

В ходе дипломной работы 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 требованиями разрабатываемого программного средства являются:

- регистрация пользователей;
- идентификация пользователей;
- аутентификация и авторизация пользователей;
- хранения данных в удаленной базе данных;
- пересылка сообщений между пользователями в режиме реального времени;
- сохранения историй чатов между пользователями 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, электронная почта, 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 иницирующем 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 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 программное средство мессенджера, установленное на устройстве пользователя. Сервер – это центральная система, которая управляет передачей сообщений между пользователями.

1.2 Обзор существующих программных средств по теме дипломной работы

В ходе дипломной работы 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 полезная функциональность 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 влечет за собой неблагоприятный пользовательский опыт, рисунок 1.1.

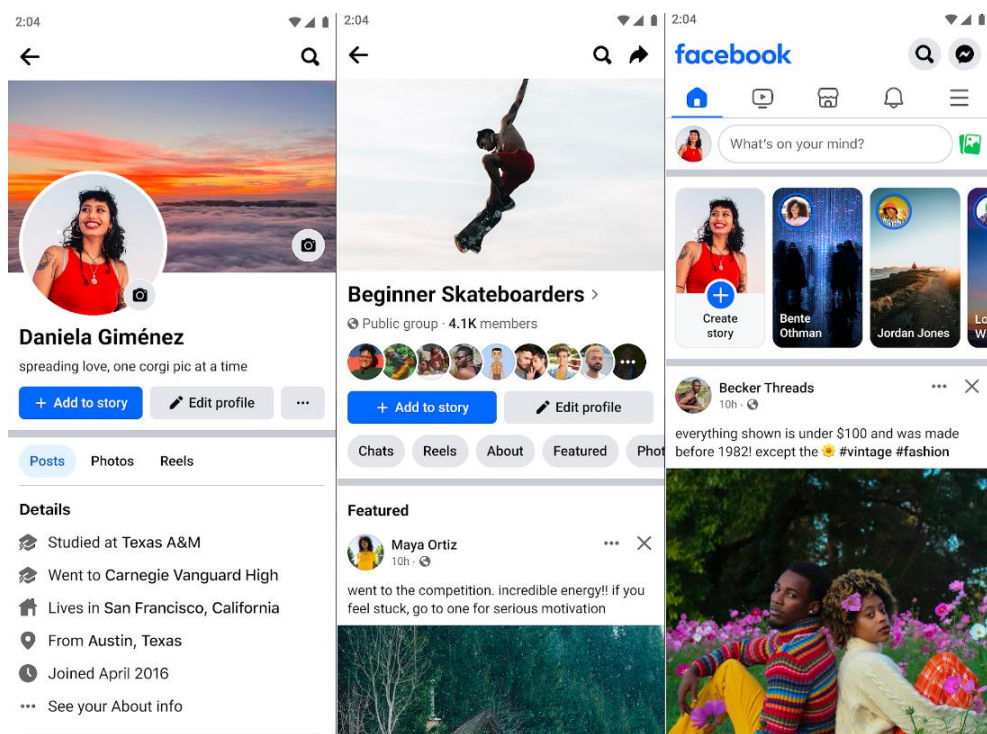


Рисунок 1.1 – Скриншот пользовательского интерфейса Facebook

Обзор онлайн мессенджера WhatsApp.

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

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:

- возможность совершать видео и аудио звонки;
- возможность 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 недостатков можно отметить что:

- отправка файлов размером максимум 100 МБ;
- временами в фоне не работает или не присылает уведомления;
- на телефонах фирмы Apple расходует излишнее количество батареи;
- наличие проблем с безопасностью 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.

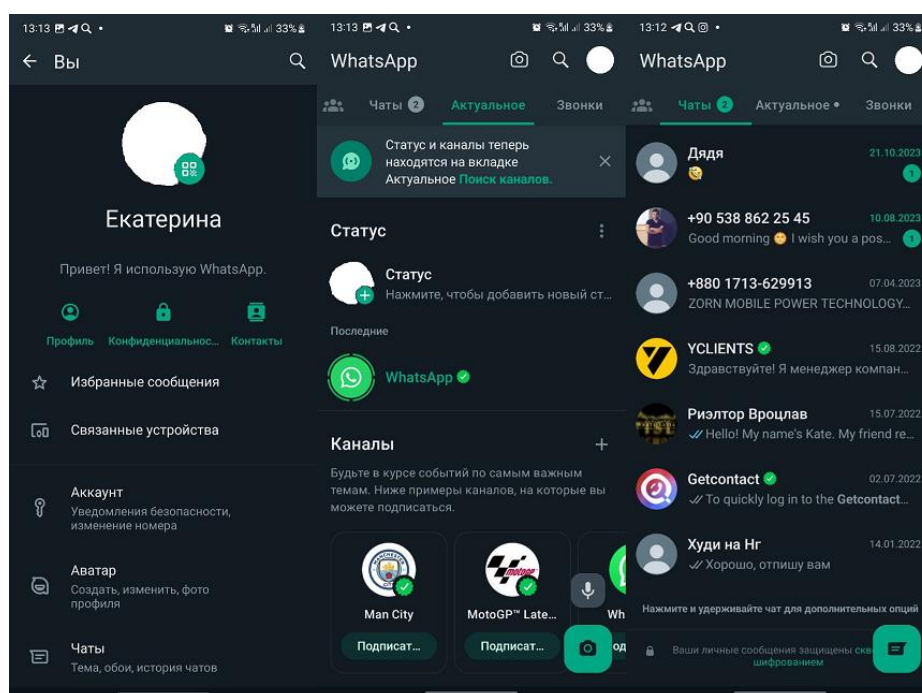


Рисунок 1.2 – Скриншот пользовательского интерфейса WhatsApp

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 [10].

Сначала работа задумывалась 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, обмениваться 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].

ВКонтакте постоянно радуется 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 приводит 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.

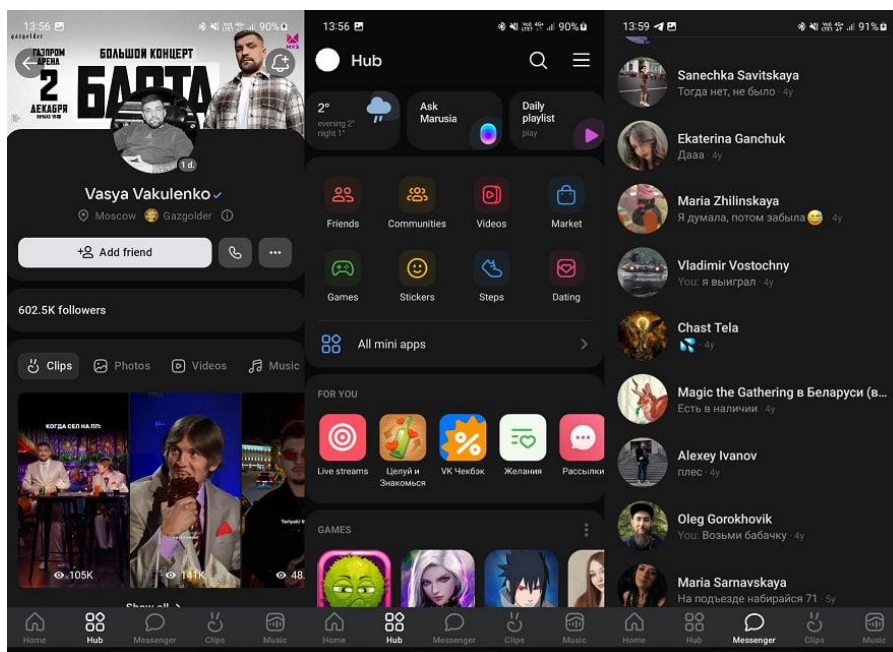


Рисунок 1.3 – Скриншот пользовательского интерфейса ВКонтакте

Обзор онлайн мессенджера Telegram.

Telegram – 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 Telegram – это отправка сообщений. Вы можете отправлять сообщения своим друзьям, семье, коллегам – кому угодно, кто также использует это приложение. Кроме того, Telegram 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, Telegram также позволяет отправлять различные файлы: фотографии, видео, аудиозаписи, документы и даже геолокацию. Вы можете делиться своими впечатлениями и воспоминаниями с другими людьми, отправлять им фотографии с праздников или путешествий, смотреть и делиться видео или музыкой. Также Telegram 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 Telegram 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.

Сегодня Telegram 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 Telegram 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 Telegram 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, Telegram 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.

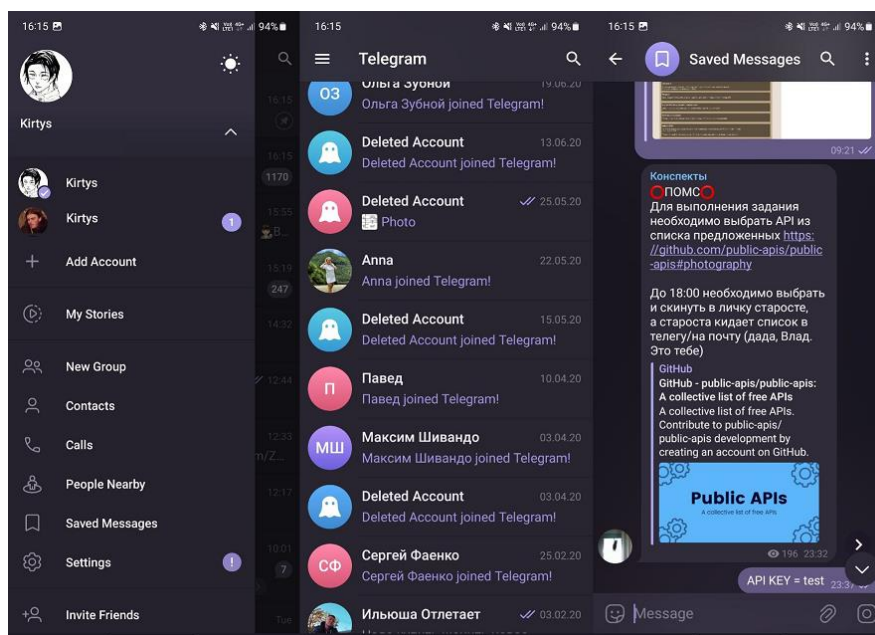


Рисунок 1.4 – Скриншот графического интерфейса телеграм

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.

Telegram 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	Whats App	ВКонтакте	Телеграм	Разрабатываемое программное средство
Авторизация	+	+	-	+	+
Внутренний магазин	+	-	+	+	-
Реклама	+	-	+	-	-
Защита данных	+	+	+	+	+
Медиа-контент	+	-	+	-	-
Просмотр истории	+	+	+	+	+

Были рассмотрены существующие аналоги разрабатываемого программного средства. Были выявлены достоинства и недостатки, рассмотрены основные функциональные возможности 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

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

Основными критериями при выборе языка программирования 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 закрывая 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 развитие ASP.NET как кроссплатформенной технологии, которая развивается как opensource-проект. Данное развитие платформы в дальнейшем получило название ASP.NET Core, собственно как ее официально именуют 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.

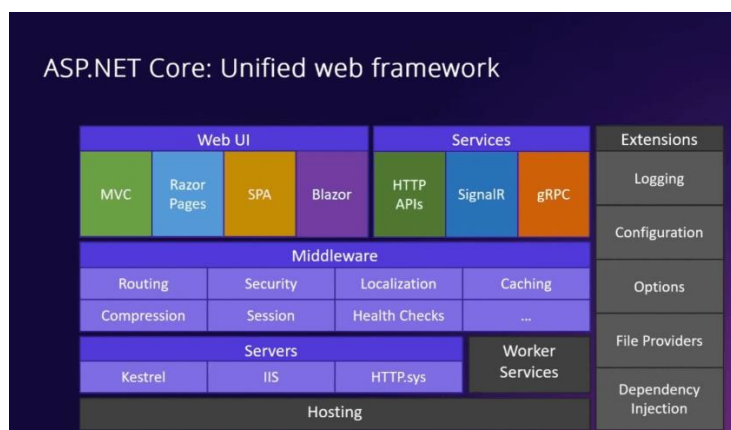


Рисунок 1.5 – Архитектура платформы ASP.NET Core

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 современного 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 обработку 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.

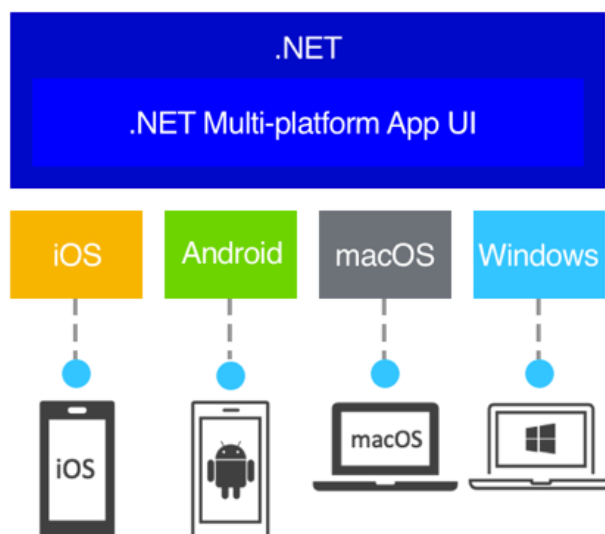


Рисунок 1.6 – Архитектура .NET MAUI

.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 giro 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;

- иницилируете 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 программном средстве 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 [16].

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.

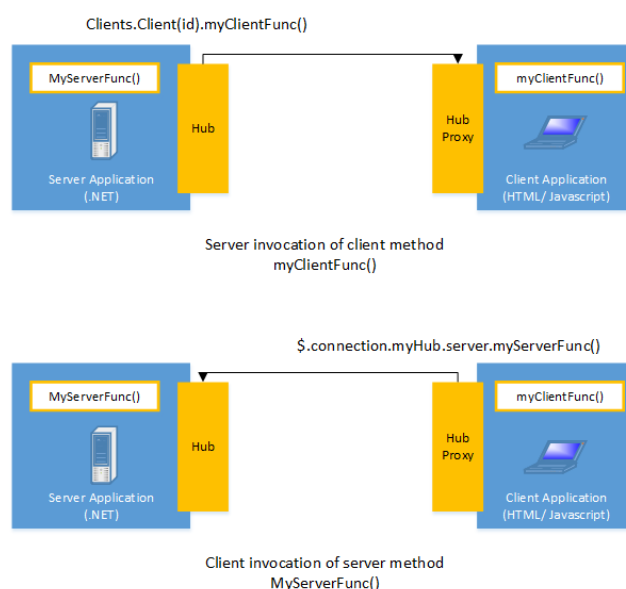


Рисунок 1.7 – Схема вызова процедур между серверами

SignalR использует новый транспорт WebSocket там, где это доступно, и при необходимости возвращается к более старым. Хотя вы, безусловно, можете написать свое программное средство 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 запрограммировать программное средство, 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 транспортном протоколе 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 дипломную работу

Реализуемое программное средство направленно на организацию внутреннего коммуникационного взаимодействия между пользователями для реализации средства по пересылки информации в режиме реального времени необходимо определить и решить следующие задачи:

- идентификация пользователя;
- авторизация и аутентификация пользователя;
- регистрация пользователя;
- хранения 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 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 которых 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 определенную функцию, рисунок 2.1.

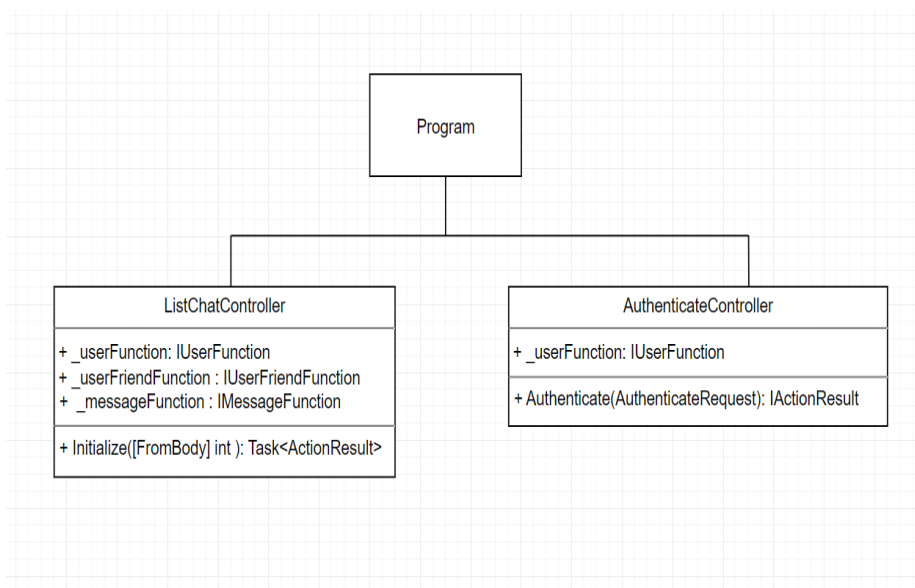


Рисунок 2.1 – Фрагмент UML диаграммы классов отвечающий за работу аутентификации и инициализацию

Класс ListChatController отвечает за вывод на экран информации о всех чатах пользователя в соответствии с информацией, полученной из базы данных. Класс реализует интерфейсы: IUserFunction, IUserFriendFunction и IMessageFunction которые реализуют методы для получения информации с базы данных. Эти методы реализованы в классах UserFunction, UserFriendFunction и MessageFunction. 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 пользователем. 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 параметров авторизации. При несоответствии введенных данных с данными в базе данных аутентификация не будет пройдена и пользователь не получит доступ в базу данных. 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 = VerifyPassword (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 работе 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 находится 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.

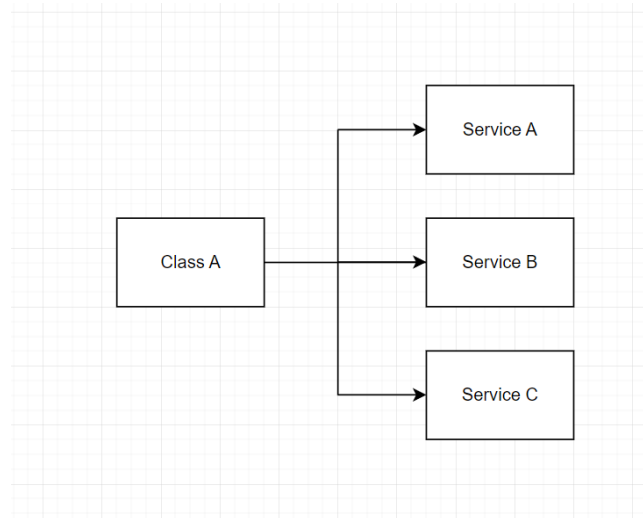


Рисунок 2.2 – Диаграмма компонентов

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 использованием внедрения зависимостей.

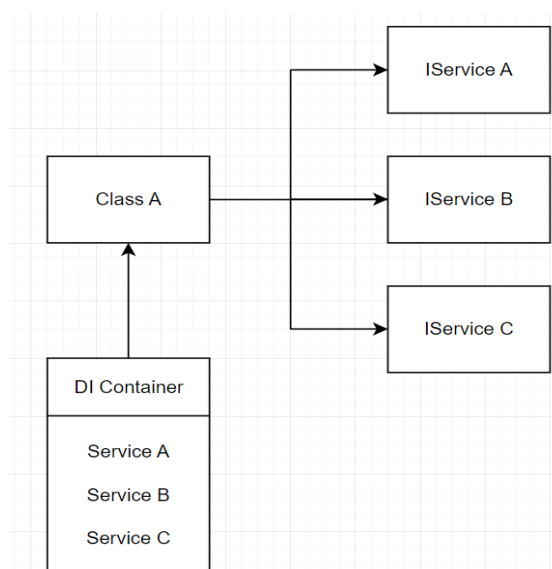


Рисунок 2.3 – Внедрения зависимости

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 builder класса WebApplication и вызвать метод CreateBuilder после чего необходимо 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;
- DELETE.

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 приложения, которое для реализации функционала которого используется Web API.

Обычно функциональные особенности, реализованные в контролерах. Для этого не обходимо помети что данный контроллер 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.

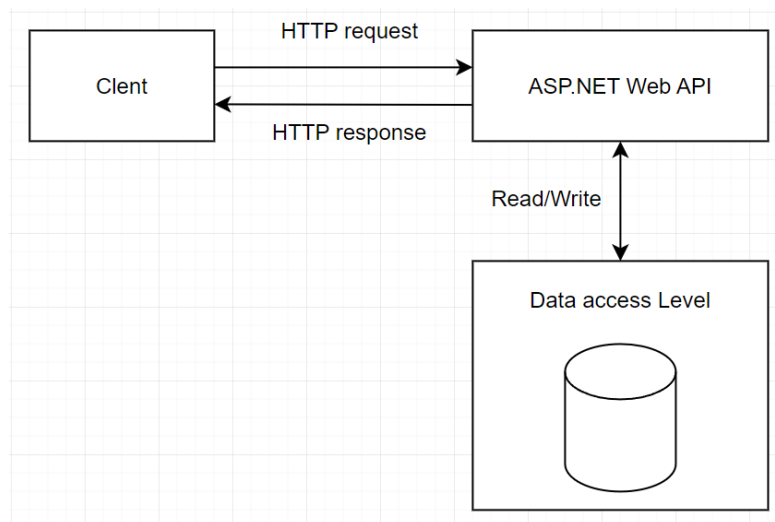


Рисунок 2.4 – Схема взаимодействие с Web API

Как видно из выше предоставленной схемы Web API находится в центре цепочки взаимодействия и является основным звеном, которое выполняет весь основной функционал по передаче, получению и работе с данными. Служит для организации работы с удаленной базой данных, 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.

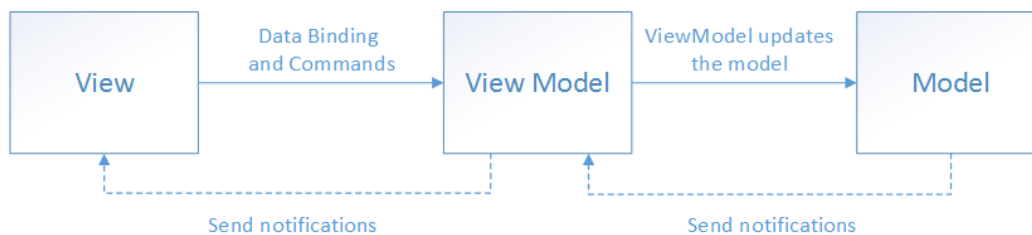


Рисунок 2.5 – Схема реализации MVVM шаблона

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. Системные требования максимально оптимизированы, что делает программное средство доступным для широкого круга пользователей, позволяя каждому установить его без проблем. Прежде чем перейти к использованию чата необходимо будет пройти простую регистрацию, а после авторизацию. В программном средстве присутствует система оповещений пользователя через шторку уведомлений.

В ходе проектирования функционального процесса было принято решение создать приватный чат между пользователями 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;
- verifyPassword – 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 имеется 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 [17].

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.

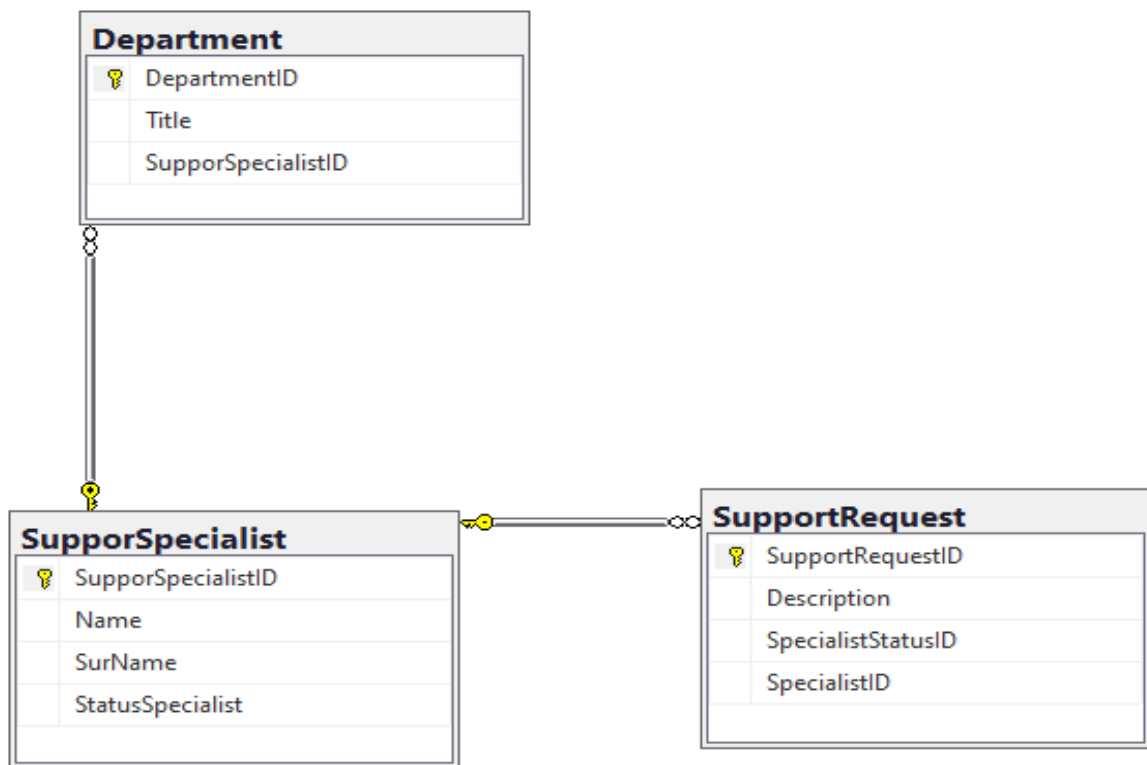


Рисунок 2.6 – Пример реализации связей в MS SQL Server

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.

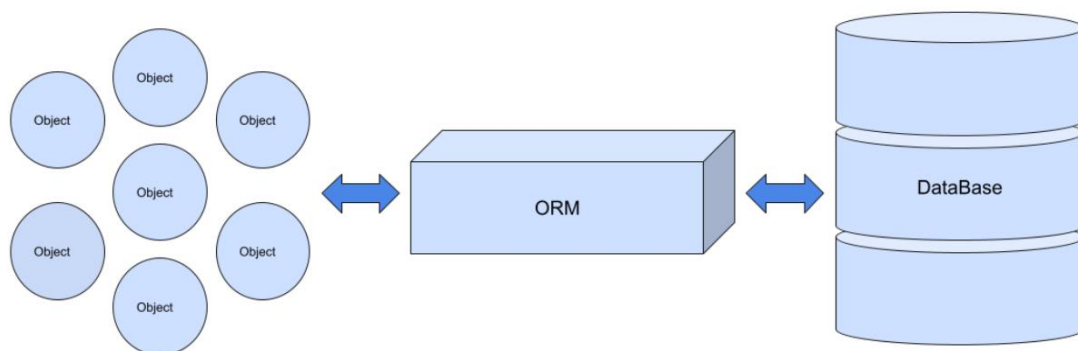


Рисунок 2.7 – Основной принцип работы ORM

Отображение объясняет, как объекты связаны с разными таблицами, используя эту информацию для преобразования данных между ними. ORM генерирует код SQL для реляционной базы данных в ответ на изменения, которые программное средство 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 синтаксический тип кода не нужно переписывать, что значительно экономит время разработки.

В данной работе мы будем использовать такую ORM как Entity Framework. Entity Framework представляет ORM-технологии 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 [18]. 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 приложениях. Изначально создаются классы на основе которых в последующем будут созданы сущности в базе данных пример такого класса выглядит следующим образом [19].

```
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 унаследоваться 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.

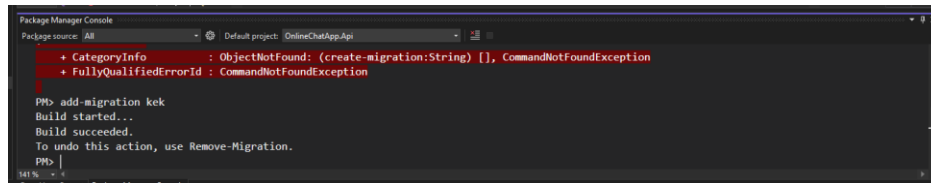


Рисунок 2.8 – Создание миграции при помощи Entity Framework

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.

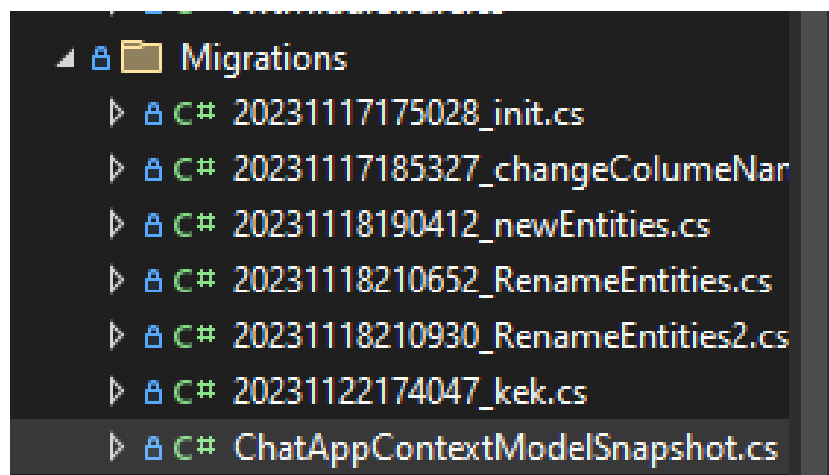


Рисунок 2.9 – Директория, автоматически созданная для миграций

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.

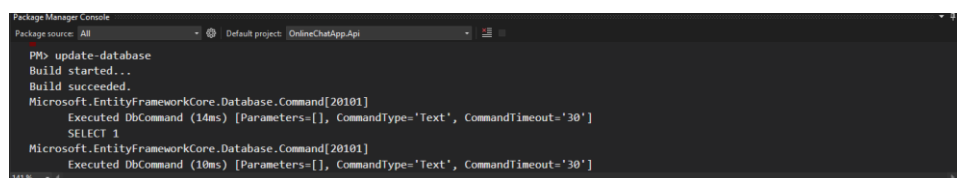


Рисунок 2.10 – Применения миграции к базе данных

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.

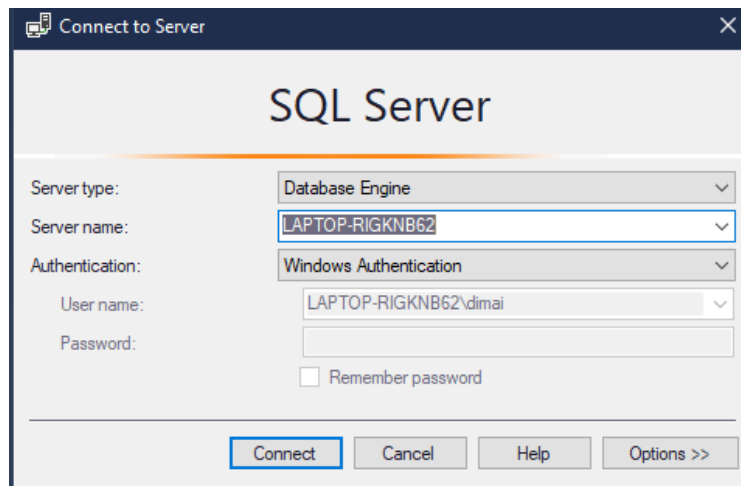


Рисунок 2.11 – Подключение к базе данных через MS SQL Server

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

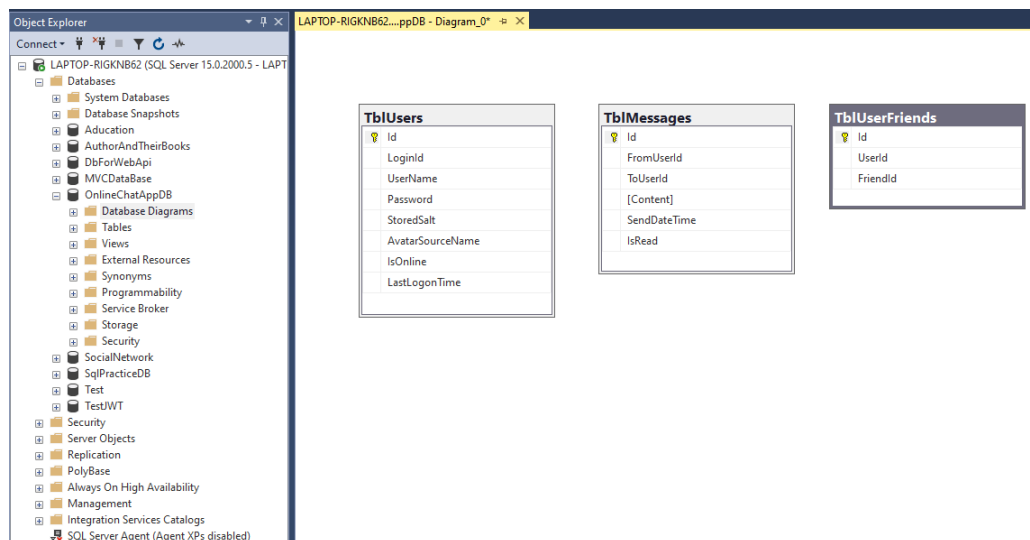


Рисунок 2.12 – Диаграмма сущностей базы данных

После всех выше описанных действий можно приступать 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 переменной выполняя 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 создании 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.

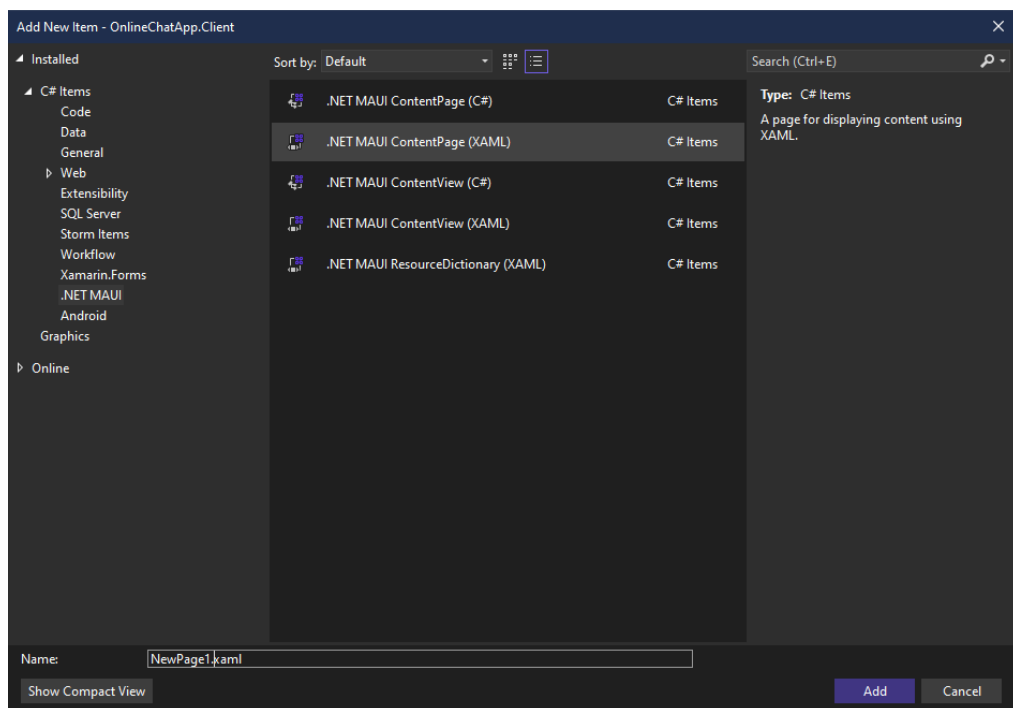


Рисунок 2.13 – Создание ContentPage в .NET MAUI

Затем Visual Studio создает новую ContentPage производную страницу, XAML код которой будет выглядеть 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 [20]. 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 результата дизайна экрана логина показан на рисунке 2.14.

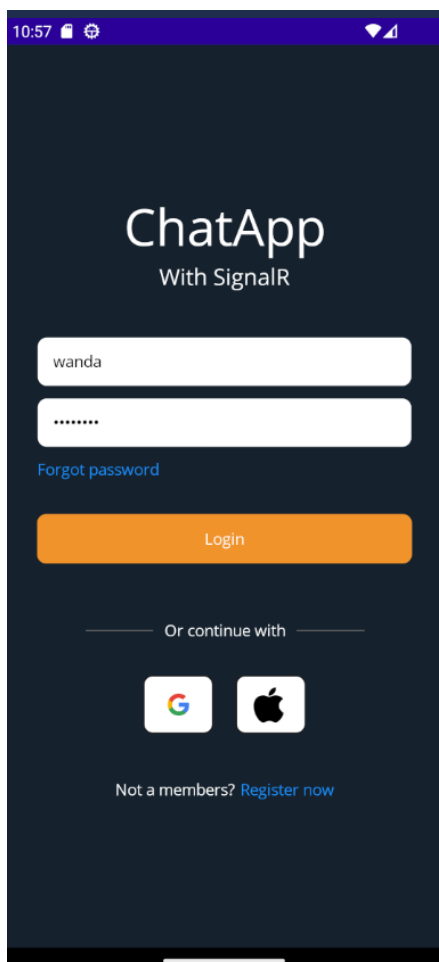


Рисунок 2.14 – Создание пользовательского интерфейса логина

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

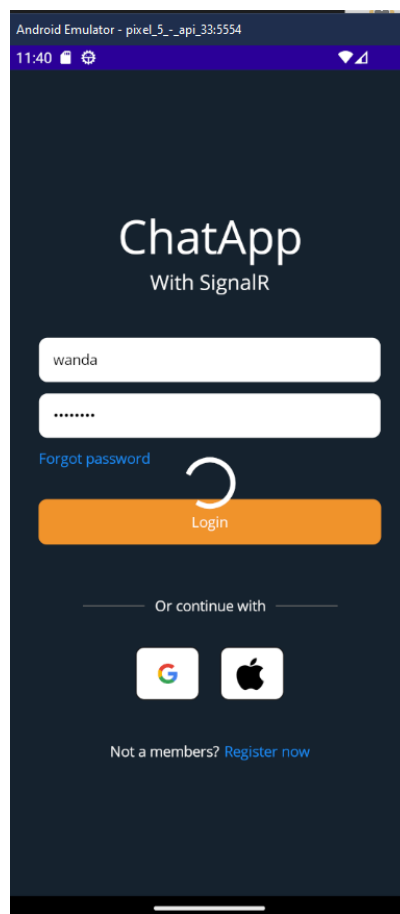


Рисунок 2.15 – Реализация `ActivityIndicator` на странице логина

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 [21].

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



Рисунок 2.17 – Реализация приватного чата

Таким образом был создан простой и интуитивной понятный 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.

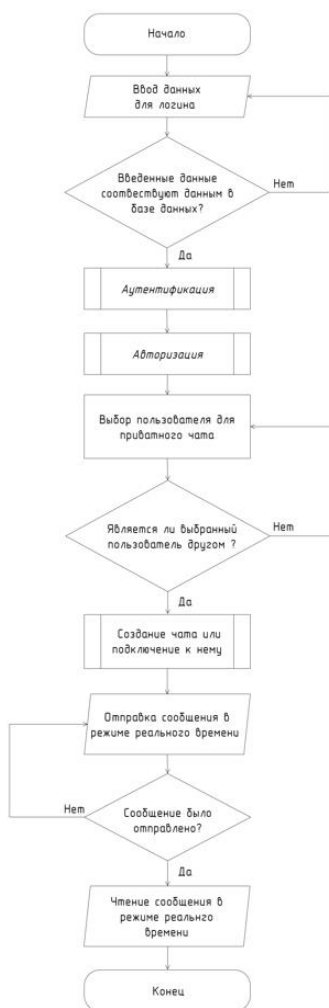


Рисунок 2.18 – Схема алгоритма работы приватного чата

Для реализации чата в режиме реального времени использовался SignalR. SignalR предоставляет простой API для создания функциональности, которая 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 функциональность, которая 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 основная функциональность, 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

```
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 названием 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 новый 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 программного средства. 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 делом необходимо сформировать `Http`-запрос для отправки на сервер методом `Post` с 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 получить 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 программного средства. 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 программное средство или вносят изменения в существующее, они могут допускать ошибки. Тестирование помогает выявить эти проблемы и убедиться, что программное средство 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 [22].

Представьте, что все эти люди объединяются, чтобы создать какой-то продукт. Они разрабатывают его, выпускают на рынок. А позже пользователи вдруг выясняют, что где-то в продукте есть ошибки. 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 котором 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 [23].

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-релизная 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 [24].

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 программного средства включает в себя компиляцию программного средства 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 программного средства 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 выявлено критических ошибок и были достигнуты следующие результаты:

- при компиляции и развертывание программного средства 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 понятную функциональность и не вызывает никаких нареканий;

- на различных версиях Android программное средства 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.

3 ОЦЕНКА КОЛИЧЕСТВЕННЫХ ПОКАЗАТЕЛЕЙ ФУНКЦИОНИРОВАНИЯ ПРОГРАММНОГО СРЕДСТВА

3.1 Оценка временных показателей программного средства

Оценка временных затрат на разработку программного обеспечения начинается с анализа требований и простирается через все этапы до внедрения готового продукта. В нынешних реалиях существует огромное множество способов и методов оценки. Тем не менее не существует способа, дающего стопроцентный результат. Отсюда следует, что для оценки временных показателей необходимо максимально точно сформулировать требования к разрабатываемому продукту. Это позволит оценить временные затраты проекта на начальных этапах с меньшей погрешностью. Оценка временных затрат завершённого продукта зависит от того, сколько этапов управления разработкой было завершено. Чем их больше, тем точнее оценка временных затрат.

При проектировании необходимо учесть скорость обработки запросов на сервере, с помощью которого будет работа онлайн чат с возможностью пересылки информации в режиме реального времени. Критически важно, чтобы информация между пользователями обрабатывалась с минимальной задержкой или вообще без нее. Так же необходимо учитывать качества и скорость интернет-соединения. Для того чтобы рассчитать возможные проблемы, связанные с работой сервера, а также скорость работы сети, необходимо сделать расчет объема трафика, передаваемого между сервером и всеми устройствами, и минимальную скорость интернет-соединения, при котором программное средство будет корректно работать.

Для определения предельной нагрузки на сервер необходимо провести симуляцию максимальной нагрузки на сервер. Для создания искусственной нагрузки существуют множество программ, такие как: Apache Jmeter, Wrk, Siege и др. Для данного сервера использовалось Windows приложения Apache Jmeter [25].

Apache Jmeter – инструмент для проведения нагрузочного тестирования, разрабатываемый Apache Software Foundation. Хотя изначально Jmeter разрабатывался как средство тестирования Web-приложений, в настоящее время он способен проводить нагрузочные тесты для Jdbc-соединений, FTP, Ldap, Soap, Jms, POP3, Imap, HTTP и TCP [26].

Преимущество Jmeter заключается в ряде факторов: это бесплатный инструмент с интуитивно понятным UI или работой из консоли, поддержкой многопоточности, расширяемостью, возможностью создания разнообразных отчётов. Многие инструменты нагрузочного тестирования используют под своей основой именно движок Jmeter. Соответственно, данная программа является одной из самых популярных у специалистов.

Для того чтобы измерить максимально возможную нагрузку на сервер, необходимо запустить сервер. После чего необходимо запустить Apache Jmeter. В данной программе прописать конечные точки, на которые будут приходить запросы. Затем необходим составить пример запросов, которые будут опрвляться на сервер, количество запросов и их периодичность, рисунок 3.1.

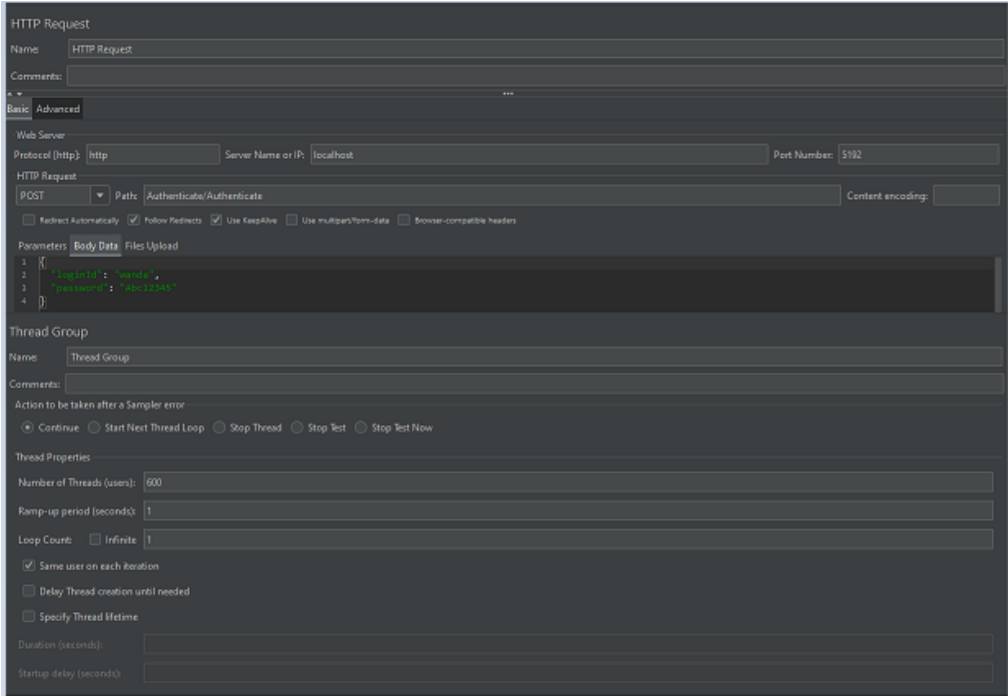


Рисунок 3.1 – Конфигурация запросов на сервер

После настройки запросов необходимо отправить их на сервер и проанализировать полученный результат после него необходимо произвести расчет сколько трафика приходится на одного пользователя, рисунок 3.2.



Рисунок 3.2 – Подробная информация об ответах сервера

После тестирования нагрузок на сервер были получены следующие данные что на один запрос пользователя на логин приходится примерно 18 пакетов один пакет равен 4 кБ в секунду. Общее количество проведенных тестов равняется 15.

$$T_1 = C \cdot D, \quad (3.1)$$

где C – общее количество пакетов, затрачиваемых одним пользователем;

D – размер одного пакета кБ.

Расчет возможного потребления трафика на одного пользователя равен:

$$T_1 = 18 \cdot 4 = 72 \text{ кБ.}$$

При анализе возможно количества пользователей получено предполагаемое число пользователей равным 40000. Расчет возможного потребления трафика на всех пользователей:

$$T_{\text{общий}} = T_1 \cdot L, \quad (3.2)$$

где T_1 – количество потребляемого трафика одним пользователем кБ;

L – количество пользователей.

Расчет возможного потребления трафика на всех пользователей равен:

$$T_{\text{общий}} = 72 \cdot 40000 = 2.88 \text{ Гб/с.}$$

По итогам расчета можно сделать вывод, что для корректной работы программного средства для пересылки информации в режиме реального времени необходимо минимальная скорость интернет-соединения. Чтобы избежать возможных проблем с работой сервера, необходимо иметь большой запас скорости. Так как при перегрузке сервера может произойти потеря или повреждения данных, что может привести к негативным оценкам пользователей.

3.2 Оценка ресурсных показателей программного средства

Оценка ресурсной эффективности заключается в измерении количественных суб-характеристик и их атрибутов: временной эффективности и используемости ресурсов комплексом программ. В стандарте ISO 25023 уделяет внимание оценке ресурсной эффективности программного обеспечения через две основные характеристики: временную эффективность и используемость ресурсов. Оценка временной эффективности подразумевает собой измерение скорости выполнения программы, время ответа и обработки запросов, посылаемых на сервер, а также производительности программы в различных условиях нагрузки.

Используемость ресурсов оценивает, насколько эффективно программа использует доступные ресурсы устройства, на котором будет разворачиваться и работать программа, такие как время обработки информации процессором, взаимодействие с памятью и эффект оказываемый на локальную память. Эти факторы определяют эффективность работы программы и её взаимодействие на систему [27].

Потребность программы в ресурсах памяти и производительности в зависимости от поставленной задачи может сильно различаться. Все это зависит от количества используемых потоков, количество необходимой памяти для работы, типа данных и их количества. Использование памяти и производительность устройства, не всегда являются определяющими для корректных функциональных задач программного средства. Однако при слишком большой нагрузке может нарушиться баланс между временем, необходимым для выполнения полного набора задач программы в реальном времени, и эффективностью выполнения этих задач. Для выявления подобных ситуаций и определения характеристик программных средств в условиях недостаточности ресурсов проводятся испытания при высокой, но допустимой интенсивности поступления исходных данных.

Данные о потребляемых программным средством ресурсов устройства. Можно определить путь запуска приложения на эмуляторе или же на реальном устройстве, но при сборе данных советуется проводить на реальных устройствах. В данном случае будет проводиться тестирование на виртуальной машине, так как программное средство не требует больших затрат ресурсов, что позволяет ограничиться виртуальным устройством [28]. Характеристики виртуального устройства равны следующим показателям:

- процессор восьмиядерный, Qualcomm Snapdragon 765G;
- разрешение экрана 1080x2340 точек;
- оперативная память 8 Гб.

Для проведения тестов затрачивая памяти при нагрузке и использование функции приложения будет использоваться основной его функционал по пересылки информации в режиме реального времени и в режиме ожидания, рисунок 3.3.

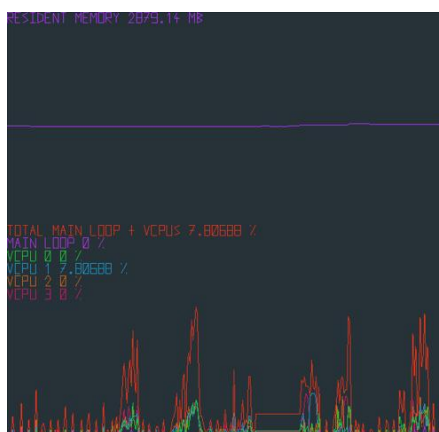


Рисунок 3.3 – Замер нагрузки на ядра процессора

После проведения тестов выявлено, что при отправке сообщения процессор получает нагрузку, равную 30 процентов на каждое ядро. В состоянии покоя 7 процентов на одно ядро. Исходя из полученных данных произведем расчет максимальной и минимальной нагрузки. Расчет максимальной нагрузки на процессор производится по формуле:

$$P_{\max} = C \cdot E_{\max}, \quad (3.3)$$

где C – количество задействованных ядер процессора;

E_{\max} – максимальная нагрузка на одно ядро.

По результатам расчета максимальная нагрузка составит:

$$P_{\max} = 8 \cdot 30\% = 240\%.$$

Таким образом, общая загрузка процессора в этом случае будет составлять 240 процентов, что может указывать на то, что все ядра процессора работают на 100% своей базовой производительности, а еще 140% – это дополнительная нагрузка или гиперпоток в многопоточных процессорах.

Расчет минимальной нагрузки на процессор производится по формуле:

$$P_{\min} = C \cdot E_{\min}, \quad (3.4)$$

где C – количество задействованных ядер процессора;

E_{\min} – минимальная нагрузка на одно ядро.

По результатам расчета минимальная нагрузка составит:

$$P_{\min} = 8 \cdot 7\% = 56\%.$$

Таким образом, общая нагрузка процессора в состоянии покоя составляет 56%. Это указывает на общую нагрузку на все ядра приложения или системы в период неактивности или минимальной активности.

Для тестирования нагрузки использовалась встроенная в эмулятор функция для отслеживания нагрузки на каждое ядро. Для тестирования программного средства была произведена отправка сообщений в режиме реального времени. После первичных замеров необходимо произвести замер постоянные [29]. Для этого необходимо произвести непрерывную отставку сообщений на протяжении минимум 5 минут. Последним этапом необходимо провести проверку нагрузки после бездействия приложения. После всех этапов проверки выявлено, что приложения работает исправно и не требует почти не какой нагрузки для его работы и будет работать на большинстве устройств и даже на устройствах старой модели.

3.3 Оценка показателей надёжности программного средства

Надёжность программного обеспечения гораздо важнее других его характеристик, например, времени исполнения. И хотя абсолютная надёжность современного программного обеспечения, по-видимому, недостижима, до сих пор не существует общепринятой меры надёжности компьютерных программ.

Вопрос обеспечения надёжности программного средства считается более важным, чем вопрос ее оценки. Ситуация выглядит парадоксальной: совершенно очевидно, что прежде чем улучшать какую-то характеристику, следует научиться ее измерять, и уж по крайней мере, необходимо иметь единицу измерения. Основная причина такого положения коренится в том, что источником ненадёжности программ служат содержащиеся в них ошибки. И если ошибки отсутствуют, то программа абсолютно надёжна. По существу, все меры по обеспечению надёжности программ направлены на то, чтобы свести к минимуму ошибки при разработке и как можно раньше их выявить и устранить после изготовления программы. Следует заметить, что безошибочные программы, конечно же, существуют, однако современные программные системы слишком велики и почти неизбежно содержат ошибки.

Под надёжностью программного средства понимают способность выполнять заданные функции, сохраняя во времени значения установленных эксплуатационных показателей в заданных пределах, соответствующих заданным режимам и условиям исполнения. Под ошибкой понимают всякое невыполнение программой заданных функций. Проявление ошибки является отказом программы [30]. Стоит выделить наиболее распространённые показатели надёжности программного средства:

- начальное число ошибок в программном средстве после сборки программы и перед ее отладкой;
- число ошибок в программном средстве, обнаруженных и оставшихся после каждого этапа отладки;
- наработка на отказ, часов;
- вероятность безотказной работы программного средства за заданное время работы;
- интенсивность отказов программного средства.

После установки приложения на тестовых устройствах и сборки программного средства никаких ошибок выявлено не было. Для развертывания приложения использовались двух разных устройствах: одно для минимальной версии Android API 26 и одно для актуальной версии Android Api 33. Так же были проведены развёртывания на устройствах разных марках, таких как Samsung, Xiaomi и Google. Для развертывания на Window, Mac и Ios использовались виртуальные машины. Ошибок так же не выявлено.

Следующим этапом необходимо пройти по всему основному функционалу:

- логин;

- регистрация;
- открытия приватного онлайн чата;
- отправка сообщений;
- получения сообщения в режиме реального времени;
- получения уведомления о последнем сообщении.

Для проверки надежной работы программного средства, исходя из временных рамок, отведенных на формирование проекта, требуется воспользоваться упрощенной оценкой надёжности программного средства. Данный метод позволит проверить надежность работы программного средства на последнем этапе разработки перед вводом в эксплуатацию. Упрощенная оценка надёжности программного средства осуществляется по формуле:

$$P = 1 - \frac{n}{N}, \quad (3.5)$$

где n – число отказов при испытании программного средства;

N – число запусков программного средства в период испытаний.

При количестве запусков программного средства в 100 было выявлено 0 сбоев в работе. Коэффициент надежности программного средства равен:

$$P = 1 - \frac{0}{100} = 1.$$

Коэффициент надежности программного средства, равный 1, указывает на исключительно высокий уровень надежности программы. Это означает, что программа работает без ошибок и сбоев в заданных условиях и прошла успешное тестирование на различных этапах разработки. Однако, необходимо помнить, что абсолютной гарантии от ошибок или сбоев программного обеспечения не существует, поэтому продолжение мониторинга и тестирования в процессе эксплуатации остается важным.

Оценка надежности программного обеспечения производится путем вычисления коэффициента ошибок на тысячу строк кода. Этот коэффициент рассчитывается с помощью следующей формулы: количество ошибок в коде делится на количество строк кода, умножается на тысячу и определяется по формуле

$$K = 1.1 \cdot \frac{E}{1000}, \quad (3.6)$$

где K – коэффициент надежности ПО;

1,1 – константа, согласно используемому языку программирования при разработке программного средства;

E – промежуточный коэффициент.

Промежуточный коэффициент E определяется по формуле:

$$E = \frac{B}{146000}, \quad (3.4)$$

где B – место, занимаемое программным средством в памяти.

Разрабатываемое программное средство занимает 8768000 бит. По результатам расчетов промежуточный коэффициент равен:

$$E = \frac{8768000}{146000} = 60,05.$$

Исходя из этого, коэффициент ошибок составляет:

$$K = 1,1 \cdot \frac{60,05}{1000} = 0,066.$$

Таким образом, коэффициент количества ошибок на 1000 строк программного кода составляет 0,066, что является приемлемым результатом для разработанного программного средства.

Так же проведено тестирования приложения в течение 24 часов на различных устройствах – это отличный способ оценить его долгосрочную работу и стабильность. В течение этого времени не наблюдалось ошибок, влияние приложения на ресурсы устройства и избыточного потребления батареи. Проведение таких нагрузочных тестов позволяет выявить проблемы, которые могут возникнуть после длительного использования приложения

Такое тестирование также помогает выявить потенциальные проблемы с утечкой памяти или другие аномалии, которые могут быть связаны с продолжительным использованием приложения. После всех выше перечисленных тестов никаких ошибок или аномалий не выявлено.

Отсутствие отказов и ошибок при работе программного обеспечения обусловлена тем, что программное средство еще не разрослось до больших масштабов, и оно еще не имеет большого количества возможностей, в котором могут возникать ошибки и привести к негативным отзывам.

4 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА

4.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию

Для ввода в эксплуатацию программного средства для пересылки информации в режиме реального времени под операционную систему Android необходимо собрать клиентское и серверное программное средство.

Первым делом необходимо открыть серверную часть в Visual Studio. Затем выбрать решение и нажать на кнопку Build для первичной сборки программного средства, рисунок 4.1.

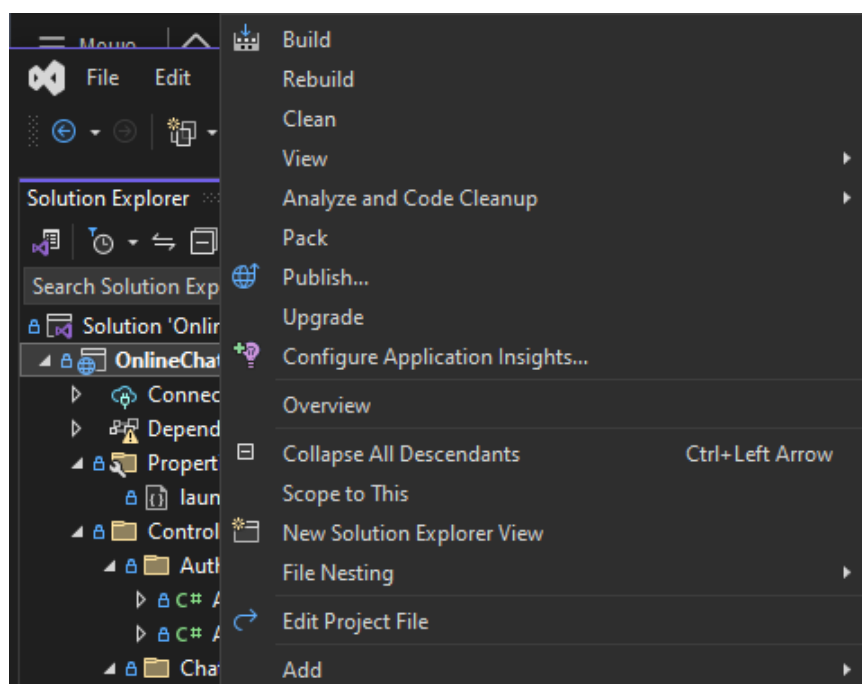


Рисунок 4.1 – Окно функций применяемых к работе

После первичной сборки необходимо проверить что сборка была произведена успешно и ошибок выявлено не было, рисунок 4.2

```
1> 17 Warning(s)
1> 0 Error(s)
1>
1>Time Elapsed 00:00:02.94
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build completed at 17:45 and took 03,861 seconds =====
```

Рисунок 4.2 – Результат первичной сборки приложения

Так как это серверная часть приложения то так же необходимо определить `ConnectionString` для подключения к базе данных и `LaunchSettings` для конфигурации сборки. Ниже представлен фрагмент файла `LaunchSettings`.

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:49841",
      "sslPort": 44386
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5192",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7032;http://localhost:5192",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Данная конфигурации позволяет запускать проект с разными локальными портами подключения для каждого определенного протокола подключения, а также для IIS веб-сервера. Данный сервер будет использовать протокол https. Для запуска сервера необходимо нажать на кнопку Start и выбрать https, рисунок 4.3.

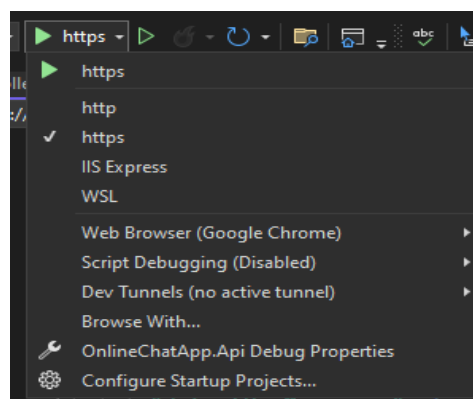


Рисунок 4.3 – Окно параметров запуска для серверной части

После запуска сервера можно переходить к запуску клиентской части для этого необходимо открыть клиентскую часть в Visual Studio. После чего необходимо так же нажать кнопку Build и выбрать там устройство, на котором будет собираться клиентская часть, рисунок 4.4.

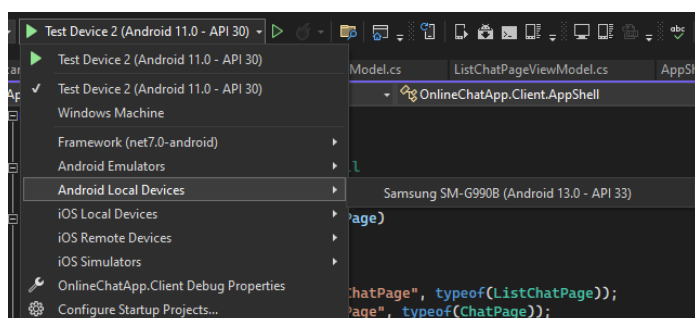


Рисунок 4.4 – Окно параметров запуска для клиентской части

После запуска сервера и клиентской части приложения процесс ввода в эксплуатацию можно считать завершенным.

Рекомендованные системные требования для Android:

- процессор восьмиядерный, Qualcomm Snapdragon 765G;
- разрешение экрана 1080x2340 точек;
- оперативная память 8 Гб.

Эти системные требования учитывают возможность тестирования программного продукта на устройстве с аналогичными характеристиками.

Рекомендованные системные требования для сервера ASP.net Web API.

- процессор шестиядерный, AMD Ryzen 5 5500;
- блок питания мощностью не менее 400 Вт;
- оперативная память 16 Гб;
- SSD-накопитель емкостью 480 Гб.

Эти системные требования позволяют проводить тестирование программного продукта на устройстве с сходными характеристиками, обеспечивая соответствие и оптимальную производительность приложения при разработке и проверке на локальной машине или тестовом сервере.

Рекомендованные системные требования для базы данных.

- процессор четырехядерный, AMD Ryzen 5 3400GE OEM;
- блок питания мощностью не менее 300 Вт;
- оперативная память 4 Гб;
- SSD-накопитель емкостью 2 Тб.

Эти системные требования учитывают возможность тестирования программного продукта на устройстве с сопоставимыми характеристиками, обеспечивая соответствие и эффективную работу баз данных при локальном тестировании на собственной машине или тестовом сервере.

4.2 Руководство по эксплуатации программным средством

При первом запуске клиента пользователь попадает на страницу логина. На этой странице находится форма для логина и переход на страницу регистрации.

Графический интерфейс страницы логина показан на рисунке 4.5.

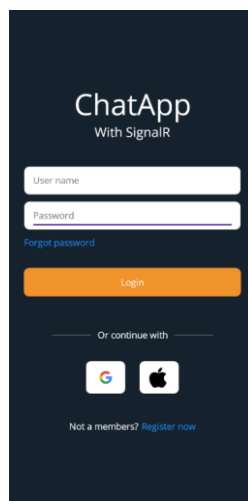


Рисунок 4.5 – Страница логина

Для логина необходимо такие данные как имя пользователя и его пароль. После чего нажать на кнопку «Login». Далее клиент отправит запрос на сервер с введенными данными обработает и отдаст ответ. Если введены не верные данные сервер даст что данные не верный и какие конкретно. Клиентская часть при получении ответа что данные не верны покажет всплывающее окно с описанием того что данные не верны, рисунок 4.6.

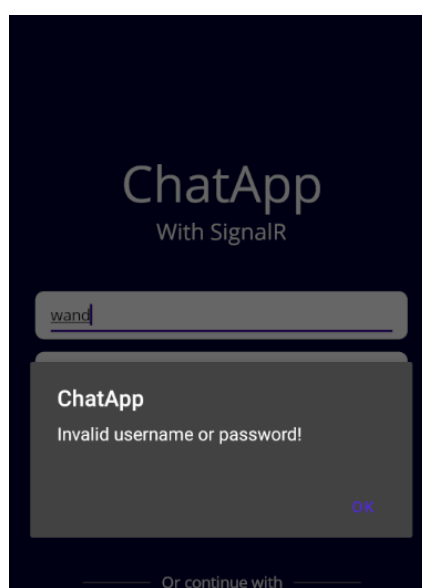
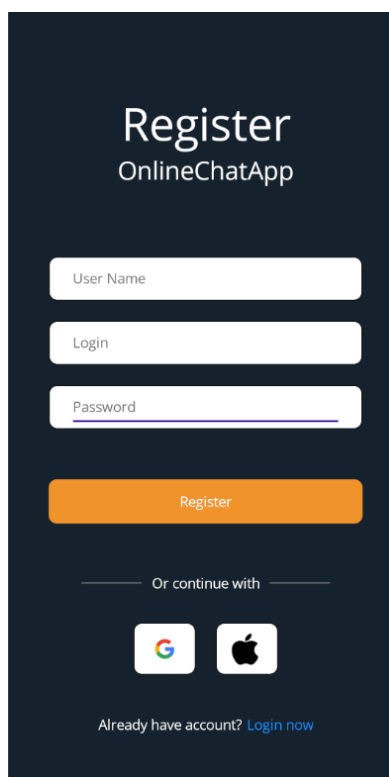


Рисунок 4.6 – Оповещение о неправильности введенных данных

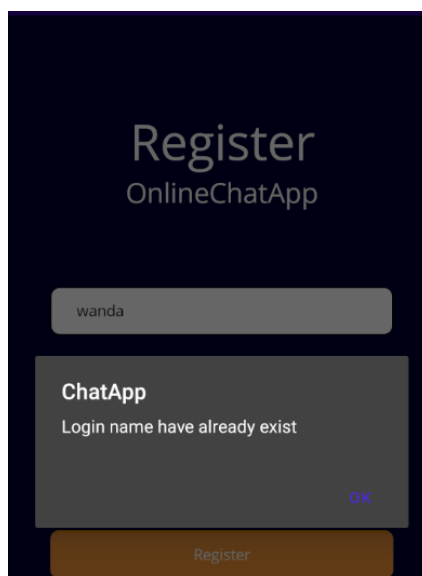
Если пользователь не зарегистрирован, то ему не обходимо нажать на кнопку «Register now» загрузится страница регистрации, которая показана на рисунке 4.7. После чего необходимо ввести все данные для регистрации такие как имя пользователя, логин пользователя и пароль.



The image shows a registration screen for 'OnlineChatApp'. It has a dark blue background. At the top, the word 'Register' is in large white font, with 'OnlineChatApp' below it in a smaller white font. There are three white input fields: 'User Name', 'Login', and 'Password'. Below these is an orange 'Register' button. Underneath the button is the text 'Or continue with' flanked by two icons: Google and Apple. At the bottom, there is a link that says 'Already have account? Login now'.

Рисунок 4.7 – Страница регистрации

Если ввести логин, который уже есть в базе система оповестит о том что этот логин уже занят которое показано на рисунке 4.8.



The image shows the same registration screen as in Figure 4.7, but with an error message. The 'Login' field now contains the text 'wanda'. A grey dialog box is overlaid on the screen with the title 'ChatApp' and the message 'Login name have already exist'. There is an 'OK' button in the bottom right corner of the dialog box. The 'Register' button at the bottom is now greyed out.

Рисунок 4.8 – Оповещение о совпадении логинов в базе данных

После успешной регистрации или логина пользователя система переходит на страницу с чатами на которой располагаются все личные чаты пользователя с друзьями, сверху список всех друзей и имя пользователя с его аватаром, рисунок 4.8.

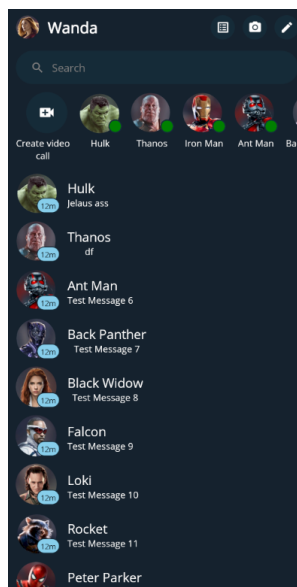


Рисунок 4.8 – Страница чатов пользователя

Далее необходимо нажать на чат, к которому пользователь хочет обратиться после чего откроется приватный чат с этим пользователем на данном экране слева расположены сообщения который пользователь получает справа же сообщения которые отправляет сам пользователь что показано на рисунке 4.9.

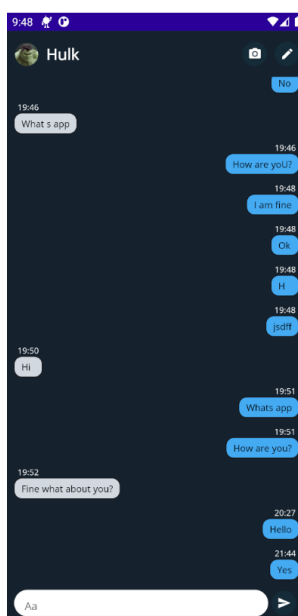


Рисунок 4.9 – Страница личного чата между пользователями

После чего необходимо вернуться на экран со всеми чатами и нажать на кнопку просмотра списка всех пользователей приложения после чего система предоставит список всех пользователей как показано рисунке 4.10.

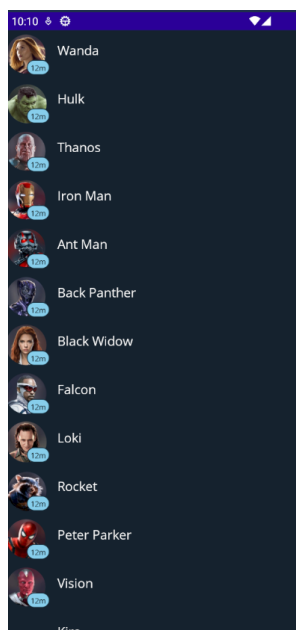


Рисунок 4.10 – Страница списка всех пользователей

Так же при получении сообщений от других пользователей система отправляет уведомления в шторку уведомления с подписью от кого и какой сообщения он отправил, рисунок 4.11.

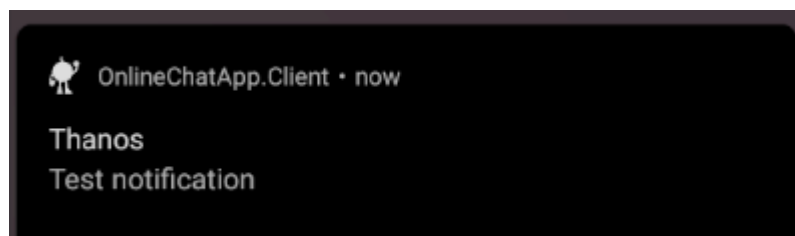


Рисунок 4.11 – Уведомления о сообщении от пользователя

Для выхода из приложения и его закрытия необходимо свернуть программное средство затем зайти в диспетчер приложений и закрыть его.

5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ПОД ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID ДЛЯ ОРГАНИЗАЦИИ РАБОТЫ ОНЛАЙН ЧАТА С ВОЗМОЖНОСТЬЮ ПЕРЕСЫЛКИ ИНФОРМАЦИИ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

5.1 Характеристика программного средства, разрабатываемого для реализации на рынке

Онлайн мессенджеры стали неотъемлемой частью нашей повседневной жизни, обеспечивая удобное и мгновенное общение в любое время и в любой точке мира. Они предлагают широкий спектр возможностей: от обмена текстовыми сообщениями в режиме реального времени до видеозвонков и совместной работы над проектами. Главное преимущество онлайн мессенджеров – это их доступность и мобильность, позволяющие людям оставаться на связи в реальном времени, будь то личные беседы, рабочие обсуждения или групповые встречи. Благодаря шифрованию и защите данных, такие платформы обеспечивают безопасность общения, что становится особенно важным в сфере бизнеса и личных переговоров.

Разрабатываемое программное средство для пересылки информации в режиме реального времени под операционную систему Android отличается легкостью использования, понятным интерфейсом, способностью передавать информацию мгновенно и обеспечивать высокий уровень защиты. Оно предоставляет простоту в управлении, интуитивно понятный дизайн, мгновенную передачу данных и гарантирует высокий уровень безопасности.

Так как технология .NET MAUI подразумевает собой разработку кроссплатформенного приложения, следовательно, распространения продукта будет проходить на все возможных платформах таких как GooglePlay, AppStore, MicrosoftStore и так далее.

Для привлечения внимания к работе будет запущена реклама в социальных сетях таких как Facebook, WhatsApp и Телеграм. Так же будет куплена реклама у такого медиа ресурса как YouTube.

В результате разработки и продвижения программного средства ожидается получить чистую прибыль от просматриваемой пользователями рекламы.

5.2 Расчет инвестиций в разработку программного средства для его реализации на рынке

Первоначально производится расчет затрат на основную заработную плату команды разработки.

Данный расчёт осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

$$Z_o = \sum_{i=1}^n Z_{чи} \cdot t_i, \quad (5.1)$$

где n – категории исполнителей, занятых разработкой программного средства;

$Z_{чи}$ – часовая заработная плата исполнителя i -й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i -й категории, определяется исходя из сложности разработки программного обеспечения и объёма выполняемых им функций, ч.

Часовая заработная плата каждого члена команды разработки рассчитывается путем деления его месячной заработной платы на количество рабочих часов в месяце в данном проекте оно равно 168 ч.

В работе над проектом задействованы следующие специалисты:

- Software engineer, с трудоемкостью в 200 часов, окладом 5000 рублей;
- UI/UX дизайнер, с трудоемкостью в 40 часов, окладом 1300 рублей;
- QA специалист, с трудоемкостью в 100 часов, окладом 2000 рублей.

Расчет затрат на основные оклады представлен в таблице 5.1

Таблица 5.1 – Расчет затрат на основную заработную плату команды разработчиков

Категория исполнения	Месячный оклад, р.	Часовой оклад, р.	Трудоемкость работ, р.	Итог, р.
Software engineer	5000	29,76	200	5952
UI/UX дизайнер	1300	7,73	40	309,2
QA специалист	2000	11,90	100	1190
Всего затрат				7451,2

Так как все дополнительные и стимулирующие выплаты работникам учтенный в сумме заработной платы, то премия отдельно не рассчитывалась.

Следующим этапом производится расчет затрат на дополнительную заработную плату команды разработчиков. Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (5.2)$$

где $З_о$ – основная заработная плата исполнителей;

$Н_д$ – норматив дополнительной заработной платы (10%).

Из расчётов, размер дополнительной заработной платы составит:

$$З_д = \frac{7451,2 \cdot 10}{100} = 745,12 \text{ р.}$$

Затем рассчитываются отчисления на социальные нужды, в фонд социальной защиты и на обязательное страхование, определяется по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (5.3)$$

где $Н_{соц}$ – норматив отчислений в ФСЗН и белгосстарх (35%).

Следовательно, размер отчислений на социальные нужды составляет:

$$Р_{соц} = \frac{(7451,2 + 754,12) \cdot 35}{100} = 2871,86 \text{ р.}$$

Прочие расходы, которые включаются в себестоимость программного средства в проценте от затрат на основную заработную плату команды разработчиков по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (5.4)$$

где $Н_{пр}$ – норматив прочих расходов (30%).

По результатам расчётов прочие расходы составят:

$$Р_{пр} = \frac{7451,2 \cdot 30}{100} = 2235,36 \text{ р.}$$

Расходы на реализацию, которые включают в себя расходы на рекламу и рассчитывается по формуле:

$$Р_p = \frac{З_о \cdot Н_p}{100}, \quad (5.5)$$

где $Н_p$ – норматив расходов на реализацию (5%).

По результатам расчётов расходы на реализацию составят:

$$P_p = \frac{7451,2 \cdot 5}{100} = 372,56 \text{ р.}$$

Итоговая сумма инвестиций на разработку рассчитывается по формуле:

$$З_p = З_o + З_d + P_{\text{соц}} + P_{\text{пр}} + P_p, \quad (5.6)$$

После расчётов всех статей затрат общая сумма затрат на разработку составит:

$$З_p = 7451,2 + 745,12 + 2871,86 + 2235,36 + 372,56 = 13676,1 \text{ р.}$$

Результаты расчетов затрат на разработку программного средства представлен в таблице 5.2

Таблица 5.2 – Затраты на разработку программного средства

Наименования статьи затрат	Расчёт по формуле (в таблице)	Сумма, р
1 Основания заработная плата разработчиков	Таблица 5.1	7451,2
2 Дополнительная заработная плата разработчиков	$З_d = \frac{7451,2 \cdot 10}{100} = 745,12 \text{ р.}$	745,12
3 Отчисления на социальные нужды	$P_{\text{соц}} = \frac{(7451,2 + 754,12) \cdot 35}{100} = 2871,86 \text{ р.}$	2871,86
4 Прочие расходы	$P_{\text{пр}} = \frac{7451,2 \cdot 30}{100} = 2235,36 \text{ р.}$	2235,36
5 Расходы на реализацию	$P_p = \frac{7451,2 \cdot 5}{100} = 372,56 \text{ р.}$	372,56
6 Общая сумма затрат на разработку	$З_p = 7451,2 + 745,12 + 2871,86 + 2235,36 + 372,56 = 13676,1 \text{ р.}$	13676,1

Как видно из таблицы 5.2, затраты на разработку программного средства под операционную систему Android для пересылки информации в режиме реального времени составляет 13676,1 р.

5.3 Оценка экономического эффекта от реализации программного средства на рынке

Экономический эффект организации-разработчика программного средства представляет собой прирост чистой прибыли величина которого зависит от объёма просмотров рекламы.

Прирост чистой прибыли, полученной разработчиком программного средства на рынке, можно рассчитать по формуле:

$$\Pi_{\text{ч}}^{\text{р}} = (\text{Ц}_{\text{отп}} \cdot N - \text{НДС}) \cdot P_{\text{пр}} \cdot \left(1 - \frac{H_{\text{п}}}{100}\right), \quad (5.7)$$

где $\text{Ц}_{\text{отп}}$ – доход от просмотра рекламы в программном средстве, р.;

$\text{Ц}_{\text{отп}} = 0,025\text{р}$ за один просмотр;

N – количество просмотров рекламы в программном средстве, за год, раз, $N_2 = 6000000$ за второй год; $N_3 = 9000000$ за третий год;

НДС – сумма налога на добавленную стоимость, р.;

$P_{\text{пр}}$ – рентабельность от просмотра рекламы, (30%);

$H_{\text{п}}$ – ставка налога на прибыль действующему законодательству, (20%).

Налог на добавленную стоимость рассчитывается по формуле:

$$\text{НДС} = \frac{\text{Ц}_{\text{отп}} \cdot N \cdot H_{\text{д.с}}}{100\% + H_{\text{д.с}}}, \quad (5.8)$$

где $H_{\text{д.с}}$ – ставка налога на добавленную стоимость в соответствии с действующим законодательством, % (20%).

На основе маркетинговых исследований потребительского спроса прогнозируемый просмотр рекламы составит около 6000000 просмотров за второй год и 9000000 за третий. Один пользователь в среднем будет просматривать рекламу около 500 раз за 1 год.

Таким образом налог на добавочную стоимость составляет:

$$\text{НДС}_2 = \frac{0,025 \cdot 6000000 \cdot 20}{100 + 20} = 25000 \text{ р.}$$

Таким образом прирост чистой прибыли при расчёте по формуле 5.7 составит:

$$\Pi_{\text{ч}_2}^{\text{р}} = (0,025 \cdot 6000000 - 25000) \cdot 0,3 \left(1 - \frac{20}{100}\right) = 30000 \text{ р.}$$

Прогнозируемая чистая прибыль за третий год составит:

$$\text{НДС}_3 = \frac{0,025 \cdot 9000000 \cdot 20}{100 + 20} = 37500 \text{ р.}$$

$$\Pi_{\text{ч}_3}^p = (0,025 \cdot 9000000 - 37500) \cdot 0,3 \left(1 - \frac{20}{100}\right) = 45000 \text{ р.}$$

5.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Расчет показателей эффективности серийного выпуска необходим для определения экономической целесообразности производства продукции. Это позволяет оценить рентабельность проекта, определить его потенциальную прибыль и срок окупаемости вложений.

Коэффициент дисконтирования рассчитывается по следующей формуле:

$$\alpha_t = \frac{1}{(1 + d)^{t-t_p}}, \quad (5.9)$$

где d – требуемая норма дисконта, которая по своему смыслу соответствует устанавливаемому инвестором желанному уровню рентабельности инвестиций, доли единиц, $d = 0,11$;

t_p – порядковый номер расчётного года, $t_p = 1$;

t – порядковый номер года, затраты и результаты которого приводятся к расчётному году.

$$\alpha_1 = 1,$$

$$\alpha_2 = \frac{1}{(1 + 0,11)^{2-1}} = 0,9,$$

$$\alpha_3 = \frac{1}{(1 + 0,11)^{3-1}} = 0,81,$$

Расчёт чистого дисконтированного дохода за четыре года реализации продукции и срока окупаемости инвестиций представлены в таблице 5.3.

Таблица 5.3 – Экономические результаты реализации программного средства

Показатель	Значения по годам расчетного периода		
	2023	2024	2025
Результат			
1 Прирост чистой прибыли, р.	0	30000	45000
2 Дисконтированный результат р. (п.1 х п.7)	0	27000	36450
Затраты			
3 Инвестиции (затраты) в реализацию проектного решения, р	13676,1	0	0
4. Дисконтированные инвестиции, р (п. 3 х п.7)	13676,1	0	0
5 Чистый дисконтированный доход по годам, р (пп. 2-4)	-13676,1	27000	36450
6 Чистый дисконтированный доход нарастающим итогом,	-13676,1	13323,9	49773,9
7 Коэффициент дисконтирования, доли единиц	1	0.90	0.81

Инвестиции в разработку окупаются на второй год реализации программного средства.

Сумма инвестиции в производство больше суммы годового прироста чистой прибыли, поэтому оценка экономической эффективности осуществляется по следующей формуле:

$$P_{II}(ARR) = \frac{\frac{1}{n} \cdot \sum_{t=1}^n P_{qt}}{\sum_{t=1}^n I_t} \cdot 100\%, \quad (5.10)$$

где n – расчетный период, лет;

P_{qt} – прирост чистой прибыли в году t в результате реализации проекта, руб;

I_t – инвестиции в году t, руб.

$$P_{II}(ARR) = \frac{\frac{1}{3} \cdot (0 + 30000 + 45000)}{13676.1} \cdot 100\% = 182.8 \ %.$$

Главным аргументом в пользу этого вывода является то, что чистый дисконтированный доход за четыре года работы производства составит 49773,9, что указывает на прибыльность проекта. Это говорит о том, что все инвестиции полностью окупаются на второй год работы приложения, что

обеспечивает быстрый возврат вложенных средств с прибылью.

Рентабельность инвестиций в проект составляет 182.8%, что представляет собой высокий показатель и подтверждает экономическую выгодность приложения. В совокупности, это делает возможным рекомендовать инвестиции в разработку программного средства для пересылки информации в режиме реального времени.

Кроме того, следует отметить, что проект обладает хорошей устойчивостью к риску. Анализ проводился с учетом разных сценариев развития событий, и даже при самых пессимистичных прогнозах проект остается прибыльным.

В целом, на основе проведенного анализа можно сделать вывод, что разработка программного средства для пересылки информации в режиме реального времени это перспективный и прибыльный проект, который заслуживает внимания инвесторов.

Делая вывод из того что рентабельность инвестиции превышает 100% и составляет 182.8%, можно с уверенностью сказать, что программное средство под операционную систему Android для пересылки информации в режиме реального времени выгодно разрабатывать и выпускать на рынок.

На основе экономических расчетов и анализа инвестиций в производство программного средства для пересылки информации в режиме реального времени можно заключить, что данный проект является экономически выгодным и заслуживает инвестиций.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы разработано программное средство под операционную систему Andorid для пересылки информации в режиме реального времени.

Были изучены исходные данные, выполнены поставленные задачи на дипломную работу, описаны функциональные возможности программного средства, проанализированы аналоги разрабатываемого продукта, выбран язык программирования для его реализации, изучены существующие аналогичные программные средства, проведен поиск готовых решений для создания данного продукта. Кроме того, осуществлено проектирование, определившее окончательную структуру программы, и разработан простой и удобный пользовательский интерфейс. Программное средство реализовано при помощи двух основных технологий это ASP.net и MAUI языка программирования C#. Для хранения данных использовалась удаленная база данных Microsoft MySQL Server.

В ходе тщательной проверки качества программного средства через функциональное тестирование была выполнена комплексная оценка его работы. Из результатов анализа следует, что не только программное средство само по себе функционирует без нареканий, но и каждая составляющая часть, каждый компонент этой системы, работает безошибочно и взаимодействует друг с другом без сбоев. Это свидетельствует о тщательном тестировании каждого аспекта программы, что в свою очередь подчеркивает высокий уровень надежности и эффективности всей разработанной системы.

В процессе работы был проведен анализ качества программного кода, который продемонстрировал высокий уровень. Кроме того, работа была охвачена планированием и тщательной проработкой его архитектуры. Оценка временных рамок и экономических аспектов указывает на то, что данная разработка является экономически целесообразной. Таким образом, результаты дипломной работы включают в себя создание программных средств, полностью отвечающих всем поставленным задачам и требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


- [1] Стивен, Л. Социальная сеть, изменившая мир: От стартапа до метавселенной / Л. Стивен. – Филадельфия. : Zerde Publishing, 2023. – 668 с.
- [2] Premiummanagement [электронный ресурс]. – Режим доступа : <https://premiummanagement.com/blog/socialnye-seti-na-rabochem-meste>.
- [3] Номейн, А. О социальных сетях и бизнесе / А. Номейн. – СПб. : Издательские решения 2018. – 59 с.
- [4] Гогохия, И. Продвижение в Telegram, WhatsApp, Skype и других мессенджерах / И. Гогохия. – М. : Бомбора, 2019. – 320 с.
- [5] Акушева, Р.Т. Ролевая Авторизация / Донской государственный технический университет: статья в журнале – научная статья. – 2020. – № 7 – 1. – С. 325 – 327.
- [6] Карпика, А.Г. Лемайкина С.В., Петрищева Е.Н. Обзор алгоритмов идентификации и аутентификации при двухфакторной авторизации // Воронежский институт Министерства внутренних дел Российской Федерации (Воронеж). – 2018. – Т. 1, № 3 (3). – С. 170 – 176.
- [7] Perfluence [электронный ресурс]. – Режим доступа : <https://perfluence.net/blog/article/zachem-nuzhen-Facebook>
- [8] Гейл, Р. Facebook. Как построить мир, который нам нужен / Р. Гейл. – СПб. : Комсомольская правда, 2020. – 112 с.
- [9] W-hatsapp [электронный ресурс]. – Режим доступа : <https://w-hatsapp.ru/preimushestva-plusy-minusy/>
- [10] Югова, А.А. Продвижение ВКонтакте / А.А Югова – СПб. : АСТ, 2021. – 330 с.
- [11] Rookee [электронный ресурс]. – Режим доступа : <https://wiki.rookee.ru/vkontakte/>
- [12] Бровкина, Е. Открой для себя социальную сеть ВКонтакте. Для новичков / Е. Бровкина. – СПб. : Издательские решения, 2019. – 390 с.
- [13] Сенаторов, А.А. Telegram. Как запустить канал, привлечь подписчиков и заработать на контенте / А.А Сенаторов – СПб. : Альпина Паблишер, 2018. – 160 с.
- [14] Васильев, А.Н. Программирование на C# для начинающих. Основные сведения / А.Н. Васильев. – М. : Бомбора, 2023. – 592 с.
- [15] Умрихин, Е.Д. Разработка веб-приложений с помощью ASP. Net Core MVC / Е.Д. Умрихин. – М. : БХВ, 2023. – 416 с.
- [16] Microsoft [электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/signalr-typescript-webpack?view=aspnetcore-7.0&tabs=visual-studio>
- [17] Болье, А. Изучаем SQL. Генерация, выборка и обработка данных / А. Болье – М. : Диалектика-Вильямс, 2021. – 400 с.
- [18] Смит, Д. Entity Framework Core в действии / Д. Смит. – М. : ДМК Пресс, 2023. – 690 с.


- [19] Шилдс, У. SQL: быстрое погружение / У. Шилдс. – СПб. : Питер, 2022. – 224 с.
- [20] Малышев, К.В. Построение пользовательских интерфейсов / К.В. Малышев – М. ДМК Пресс, 2021. – 268 с.
- [21] Акулич, М. Дизайн пользовательского интерфейса и дизайн UI/UX в разработке мобильных приложений / М. Акулич. – СПб. : Ridero, 2023. – 176 с.
- [22] Мартин, Р. Идеальная работа. Программирование без прикрас / Р. Мартин. – СПб. : Питер, 2022. – 385 с.
- [23] Бубнов, А.А. Тестирование программного обеспечения. / А.А. Бубнов. – СПб. : КУРС, 2023. – 128 с.
- [24] Atlassian [электронный ресурс]. – Режим доступа : <https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing>
- [25] Apache [электронный ресурс]. – Режим доступа : <https://jmeter.apache.org/>
- [26] Поллард, Б. HTTP/2 в действии / Б. Поллард. – М. : ДМК Пресс, 2021. – 424 с.
- [27] Яна, Т. Безопасность веб-приложений. Исчерпывающий гид для начинающих разработчиков / Т. Яна. – СПб. : Эксмо, 2023. – 464 с.
- [28] Колисниченко, Д.Н. Программирование для Android / Д.Н. Колисниченко. – М. : БХВ, 2020. – 288 с.
- [29] Умрихин, Е.Д. Разработка Android-приложений на C# с использованием Xamarin с нуля / Е.Д. Умрихин. – М. : БХВ – 2021. – 304 с.
- [30] Test Engineer [электронный ресурс]. – Режим доступа : <https://testengineer.ru/bolshoj-gajd-po-testirovaniyu-android-prilozhenij/>

ПРИЛОЖЕНИЕ А


(обязательное)

Отчет о проверке на заимствования в системе «Антиплагиат»

**АНТИПЛАГИАТ**
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ


 Участник

ТАРИФ

Free 


ИЗМЕНИТЬ

ПРОВЕРКИ

1 в 6 минут 


ПРОВЕРИТЬ ДОКУМЕНТ

ПОЛЬЗОВАТЕЛЬ

 dima.icigo@yandex.ru

ВОЙТИ В КАБИНЕТ

МЕНЮ

ru 

ГЛАВНАЯ / КАБИНЕТ / РЕЗУЛЬТАТЫ ПРОВЕРКИ

Оригинальность87,78%

Совпадения12,22%



Цитирования0%



Самоцитирования0%



ПОЛНЫЙ ОТЧЕТ


КРАТКИЙ ОТЧЕТ


ИСТОРИЯ ОТЧЕТОВ


 РАСПЕЧАТАТЬ 


 ВЫГРУЗИТЬ 


 СОЗДАТЬ ССЫЛКУ 


 Свойства документа

 Структура документа

 Текстовые метрики **NEW**

 Параметры проверки

 Статистика по документу

Авторы документа 

Еленевич

Дмитрий Александрович

Имя исходного файла

Дипломный проект Еленевич Дмитрий Александрович для антиплагиата.pdf

Название документа

Дипломный проект Еленевич Дмитрий Александрович для антиплагиата

Тип документа

Дипломная работа

РЕДАКТИРОВАТЬ СВОЙСТВА

Рисунок А.1 – Результат проверки на заимствование в системе «Антиплагиат»

ПРИЛОЖЕНИЕ Б (обязательное) Листинги программного кода

Листинг интерфейса IUserFunction

```
namespace OnlineChatApp.Api.Functions.User
{
    public interface IUserFunction
    {
        User? Authenticate(string loginId, string password);
        User GetUserById(int id);
        TblUser Register(string userName, string loginId, string password);
        Task<IEnumerable<User>> GetMembers();
    }
}
```

Листинг класса UserFunction

```
namespace OnlineChatApp.Api.Functions.User
{
    public class UserFunction : IUserFunction
    {
        private readonly ChatAppContext _chatAppContext;

        public UserFunction(ChatAppContext chatAppContext)
        {
            _chatAppContext = chatAppContext;
        }

        public User? Authenticate(string loginId, string password)
        {
            try
            {
                var entity =
                    _chatAppContext.TblUsers.SingleOrDefault(x => x.LoginId == loginId);

                if (entity == null) return null;

                var isPasswordMatched =
                    VerifyPassword(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;
            }
        }

        public TblUser Register(string userName, string loginId, string
password)
```

```

        {
            var existingUser = _chatAppContext.TblUsers.SingleOrDefault(x =>
x.LoginId == loginId);

            if (existingUser != null)
            {
                return null;
            }

            var passwordHelper = new PasswordHelper();
            var salt = passwordHelper.GenerateSalt();
            var hashedPassword = passwordHelper.HashPassword(password,
salt);

            var newUser = new TblUser
            {
                UserName = userName,
                LoginId = loginId,
                Password = hashedPassword,
                AvatarSourceName = "Test",
                StoredSalt = salt,
                IsOnline = false,
                LastLogonTime = DateTime.Now
            };

            _chatAppContext.TblUsers.Add(newUser);
            _chatAppContext.SaveChanges();

            Authenticate(loginId, password);

            return newUser;
        }

public User GetUserById(int id)
{
    var entity = _chatAppContext.TblUsers
        .Where(x => x.Id == id)
        .FirstOrDefault();
    if (entity == null) return new User();

    return new User
    {
        UserName = entity.UserName,
        Id = entity.Id,
        AvatarSourceName = entity.AvatarSourceName,
        AwayDuration = "",
        IsOnline = entity.IsOnline,
        LastLogonTime = entity.LastLogonTime
    };
}

public async Task<IEnumerable<User>> GetMembers()
{
    var tblUsers = await _chatAppContext.TblUsers.ToListAsync();

    if (tblUsers == null) return new List<User>();

    var allUsers = tblUsers.Select(tblUsers => new User
    {
        UserName = tblUsers.UserName,
        Id = tblUsers.Id,
        AvatarSourceName = tblUsers.AvatarSourceName,
        AwayDuration = "",

```

```

        IsOnline = tblUsers.IsOnline,
        LastLogonTime = tblUsers.LastLogonTime
    });

    return allUsers;
}

private bool VerifyPassword(string enteredPassword, byte[]
storedSalt, string storedPassword)
{
    string encryptedPassword =
Convert.ToBase64String(KeyDerivation.Pbkdf2(
    password: enteredPassword,
    salt: storedSalt,
    prf:KeyDerivationPrf.HMACSHA1,
    iterationCount:10000,
    numBytesRequested: 256 / 8));

    return encryptedPassword.Equals(storedPassword);
}

private string GenerateJwtToken(TblUser user)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes("1234567890123456");
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new[] { new Claim("id",
user.Id.ToString()) },
        Expires = DateTime.Now.AddDays(1),
        SigningCredentials = new SigningCredentials(
            new SymmetricSecurityKey(key),
            SecurityAlgorithms.HmacSha256Signature)
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
}
}

```

Листинг класса AuthenticateController

```

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);
    }
}

```

Листинг класса RegisterController

```

using Microsoft.AspNetCore.Mvc;

namespace OnlineChatApp.Api.Controllers.Register
{
    [ApiController]
    [Route("[controller]")]
    public class RegisterController : Controller
    {
        private IUserFunction _userFunction;

        public RegisterController(IUserFunction userFunction)
        {
            _userFunction = userFunction;
        }

        [HttpPost("Register")]
        public IActionResult Register(RegisterRequest request)
        {
            var registerResponse =
                _userFunction.Register(request.UserName, request.LoginId, request.Password);
            if (registerResponse == null)
            {
                return BadRequest(new { StatusMessage = "Login name
have already exist" });
            }

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

            return Ok(authResponse);
        }
    }
}

```

Листинг интерфейса IMessageFunction

```

namespace OnlineChatApp.Api.Functions.Message
{
    public interface IMessageFunction
    {
        Task<IEnumerable<LastestMessage>> GetLastestMessage(int userId);

        Task<IEnumerable<Message>> GetMessages(int fromUserId, int
toUserId);
    }
}

```

```

        Task<int> AddMessage(int fromUserId, int toUserId, string
message);
    }
}

```

Листинг класса MessageFunction

```

namespace OnlineChatApp.Api.Functions.Message
{
    public class MessageFunction : IMessageFunction
    {
        ChatAppContext _chatAppContext;
        IUserFunction _userFunction;

        public MessageFunction(ChatAppContext chatAppContext, IUserFunction
userFunction)
        {
            _chatAppContext = chatAppContext;
            _userFunction = userFunction;
        }

        public async Task<int> AddMessage(int fromUserId, int toUserId,
string message)
        {
            var entity = new TblMessage
            {
                FromUserId = fromUserId,
                ToUserId = toUserId,
                Content = message,
                SendDateTime = DateTime.Now,
                IsRead = false
            };

            _chatAppContext.TblMessages.Add(entity);

            var result = await _chatAppContext.SaveChangesAsync();

            return result;
        }

        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;
}

public async Task<IEnumerable<Message>> GetMessages(int
fromUserId, int toUserId)
{
    var entities = await _chatAppContext.TblMessages
        .Where(x => (x.FromUserId == fromUserId && x.ToUserId
== toUserId)
|| (x.FromUserId == toUserId && x.ToUserId
== fromUserId))
        .OrderBy(x => x.SendDateTime)
        .ToListAsync();

    return entities.Select(x => new Message
    {
        Id = x.Id,
        Content = x.Content,
        FromUserId =x.FromUserId,
        ToUserId =x.ToUserId,
        SendDateTime = x.SendDateTime,
        IsRead = x.IsRead,
    });
}
}
}

```

Листинг интерфейса IUserFriendFunction

```

namespace OnlineChatApp.Api.Functions.UserFrined
{
    public interface IUserFriendFunction
    {
        Task<IEnumerable<User.User>> GetListUserFriend(int userId);
    }
}

```

Листинг класса UserFriendFunction

```

namespace OnlineChatApp.Api.Functions.UserFrined
{
    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;
        }
    }
}

```

Листинг класса ListChatController

```

using OnlineChatApp.Api.Functions.Message;
using OnlineChatApp.Api.Functions.UserFrined;

namespace OnlineChatApp.Api.Controllers.ListChat
{
    [ApiController]
    [Route("[controller]")]
    [Authorize]
    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);
        }
    }
}

```

Листинг класса MemberController

```
using Microsoft.AspNetCore.Mvc;

namespace OnlineChatApp.Api.Controllers.Member
{
    [ApiController]
    [Route("[controller]")]
    [Authorize]
    public class MemberController : Controller
    {
        private IUserFunction _userFunction;

        public MemberController(IUserFunction userFunction)
        {
            _userFunction = userFunction;
        }

        [HttpPost("Members")]
        public async Task<ActionResult> GetAllMembers([FromBody] int
userId)
        {
            var response = new MemberResponse
            {
                User = _userFunction.GetUserById(userId),
                AllMembers = await _userFunction.GetMembers()
            };
            return Ok(response);
        }
    }
}
```

Листинг класса MessageContoroller

```
using Microsoft.AspNetCore.Mvc;
using OnlineChatApp.Api.Functions.Message;

namespace OnlineChatApp.Api.Controllers.Message
{
    [ApiController]
    [Route("[controller]")]
    [Authorize]
    public class MessageController : Controller
    {
        IMessageFunction _messageFunction;
        IUserFunction _userFunction;

        public MessageController(IMessageFunction messageFunction,
IUserFunction userFunction)
        {
            _messageFunction = messageFunction;
            _userFunction = userFunction;
        }

        [HttpPost("Initialize")]
        public async Task<ActionResult> Initialize([FromBody]
MessageInitializeRequest request)
        {
            var response = new MessageInitializeResponse
            {

```

```

        FriendInfo =
        _userFunction.GetUserById(request.ToUserId),
        Messages = await
        _messageFunction.GetMessages(request.FromUserId, request.ToUserId),

        };
        return Ok(response);
    }
}

```

Листинг класса PasswordHelper

```

using BCrypt.Net;
using System;
using System.Security.Cryptography;
using Microsoft.AspNetCore.Cryptography.KeyDerivation;

namespace OnlineChatApp.Api.Helpers
{
    public class PasswordHelper
    {
        public byte[] GenerateSalt()
        {
            byte[] salt = new byte[16];
            using (var rng = new RNGCryptoServiceProvider())
            {
                rng.GetBytes(salt);
            }
            return salt;
        }

        public string HashPassword(string password, byte[] salt)
        {
            string encryptedPassword =
            Convert.ToBase64String(KeyDerivation.Pbkdf2(
                password: password,
                salt: salt,
                prf: KeyDerivationPrf.HMACSHA1,
                iterationCount: 10000,
                numBytesRequested: 256 / 8));
            return encryptedPassword;
        }
    }
}

```

Листинг класса JwtMiddleware

```

using System.IdentityModel.Tokens.Jwt;
using System.Text;

namespace OnlineChatApp.Api.Helpers
{
    public class JwtMiddleware
    {
        private readonly RequestDelegate _next;

        public JwtMiddleware(RequestDelegate next)
        {
            _next = next;
        }
    }
}

```

```

    }

    public async Task Invoke(HttpContext context, IUserFunction
userFunction)
    {
        var token =
context.Request.Headers["Authorization"].FirstOrDefault()?.Split(" ").Last();
        if (token == null)
            token =
context.Request.Headers["ChatHubBearer"].FirstOrDefault()?.Split(" ").Last();
        if (token != null)
            AttachUserToContext(context, userFunction, token);

        await _next(context);
    }

    private void AttachUserToContext(HttpContext context,
IUserFunction userFunction, string token)
    {
        try
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes("1234567890123456");
            tokenHandler.ValidateToken(token, new
TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(key),
                ValidateIssuer = false,
                ValidateAudience = false,
                ClockSkew = TimeSpan.Zero,
            }, out SecurityToken validatedToken);

            var jwtToken = (JwtSecurityToken)validatedToken;
            var userId = int.Parse(jwtToken.Claims.First(x =>
x.Type == "id").Value);

            context.Items["User"] =
userFunction.GetUserById(userId);
        }
        catch
        {
        }
    }
}

```

Листинг класса UserOperator

```

namespace OnlineChatApp.Api.Helpers
{
    public class UserOperator
    {
        IHttpContextAccessor _httpContext;

        public UserOperator (IHttpContextAccessor httpContext)
        {
            _httpContext = httpContext;
        }
    }
}

```

```

        public User GetRequestUser()
        {
            if (_httpContext == null)
                return null;

            return _httpContext.HttpContext?.Items["User"] as User;
        }
    }
}

```

Листинг класса AuthorizeAttribute

```

namespace OnlineChatApp.Api.Helpers
{
    public class AuthorizeAttribute : Attribute, IAuthorizationFilter
    {
        public void OnAuthorization(AuthorizationFilterContext context)
        {
            var user = context.HttpContext?.Items["User"] as User;
            if (user == null)
            {
                context.Result = new JsonResult(new { StatusMessage =
"Unauhorized" });
            }
        }
    }
}

```

Листинг класса ChatHub

```

using OnlineChatApp.Api.Functions.Message;

namespace OnlineChatApp.Api.Controllers.ChatHub
{
    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 SendMessageToAll(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user,
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);
        }
    }
}

```

Листинг класса ChatAppContext

```

namespace OnlineChatApp.Api.Entities
{
    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!;
    }
}

```

ПРИЛОЖЕНИЕ В **(обязательное)** **Графический материал, поясняющий разработанное программное средство**

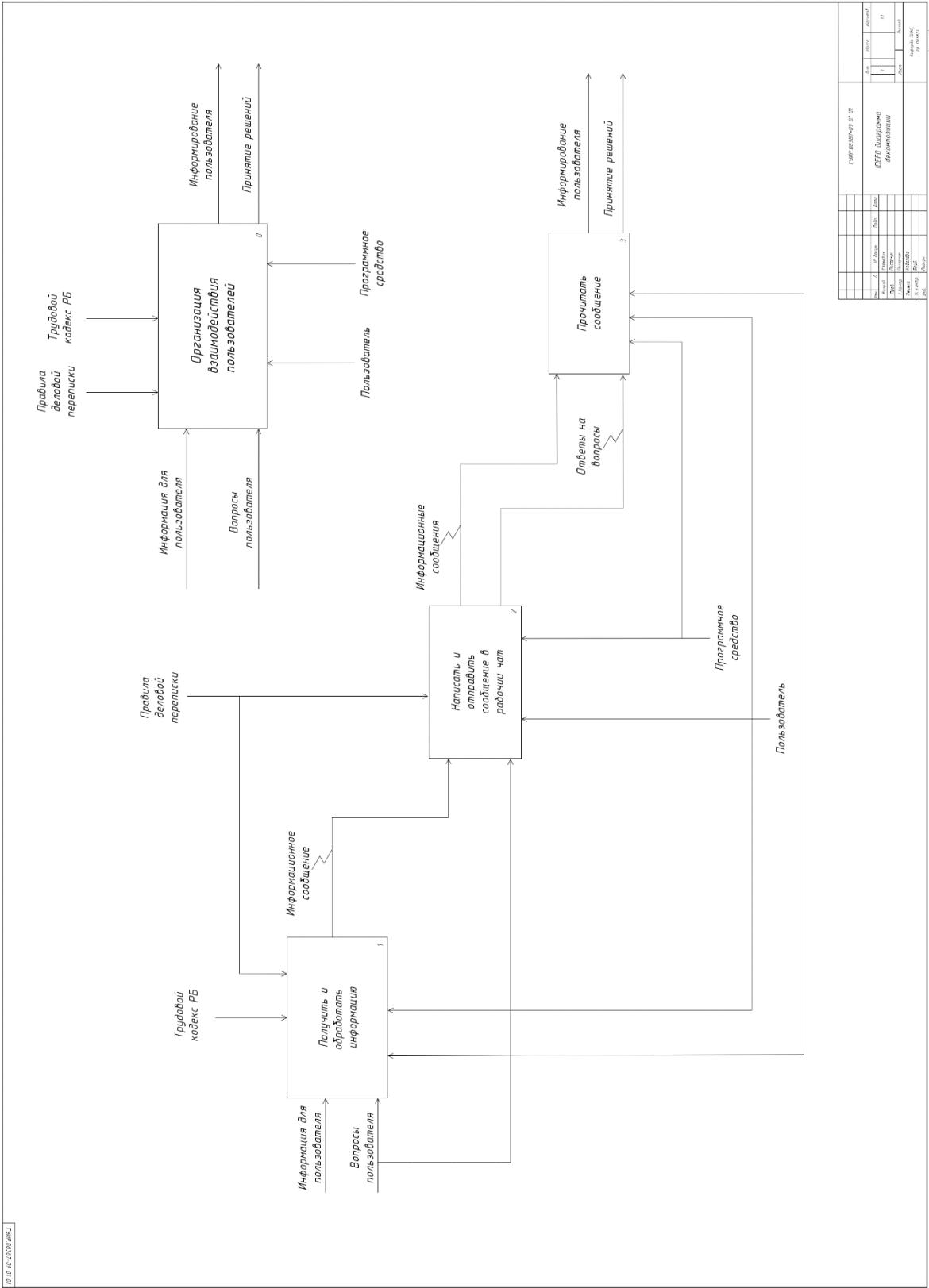


Рисунок В.1 – IDEF0 диаграмма декомпозиции

UML диаграмма вариантов использования

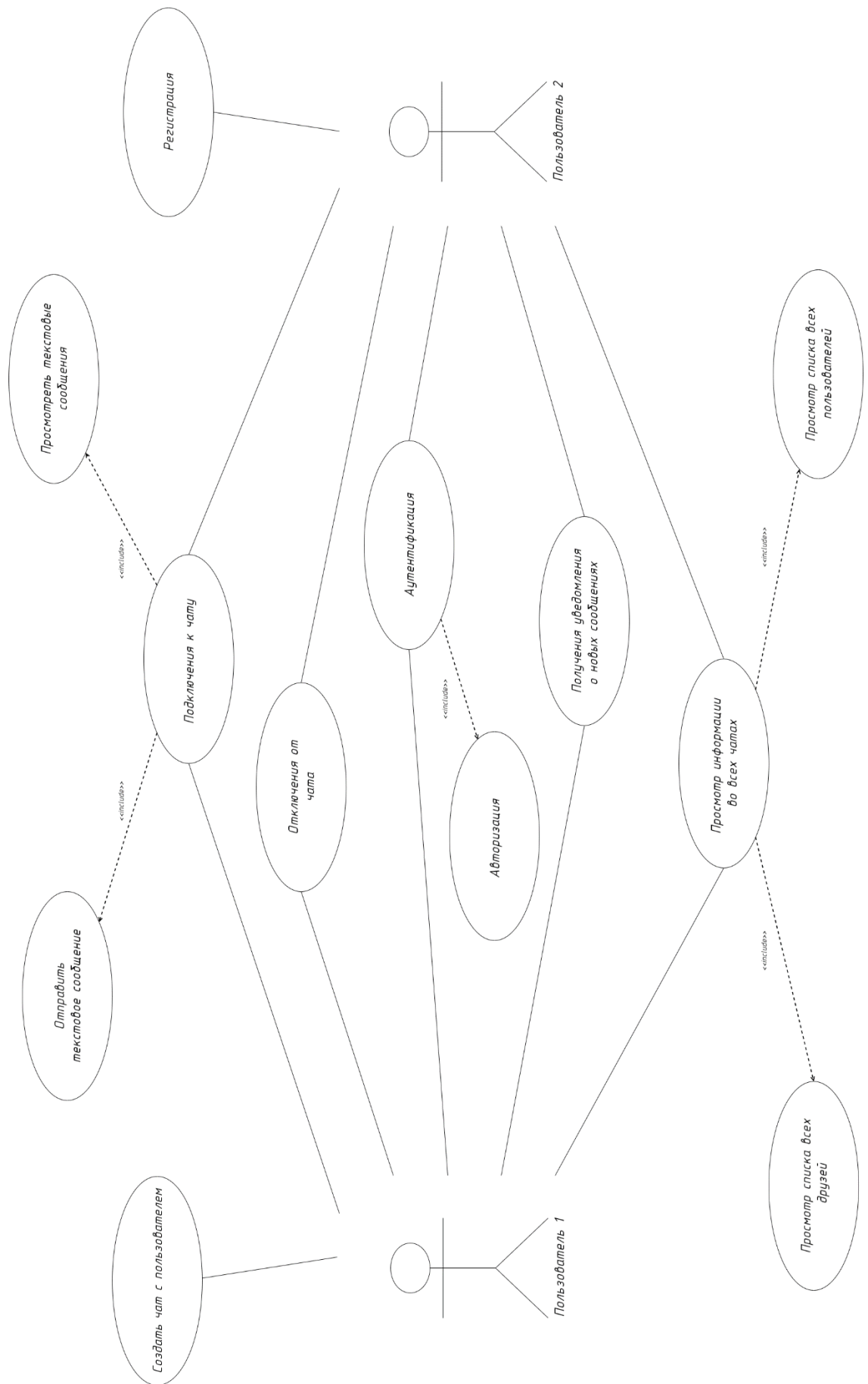


Рисунок В.4 – UML диаграмма вариантов использования

Пользовательский интерфейс программного средства



Рисунок 1 – Страница логина

Рисунок 2 – Страница регистрации

Рисунок 3 – Страница окна валидации логина

Рисунок 4 – Страница окна валидации регистрации

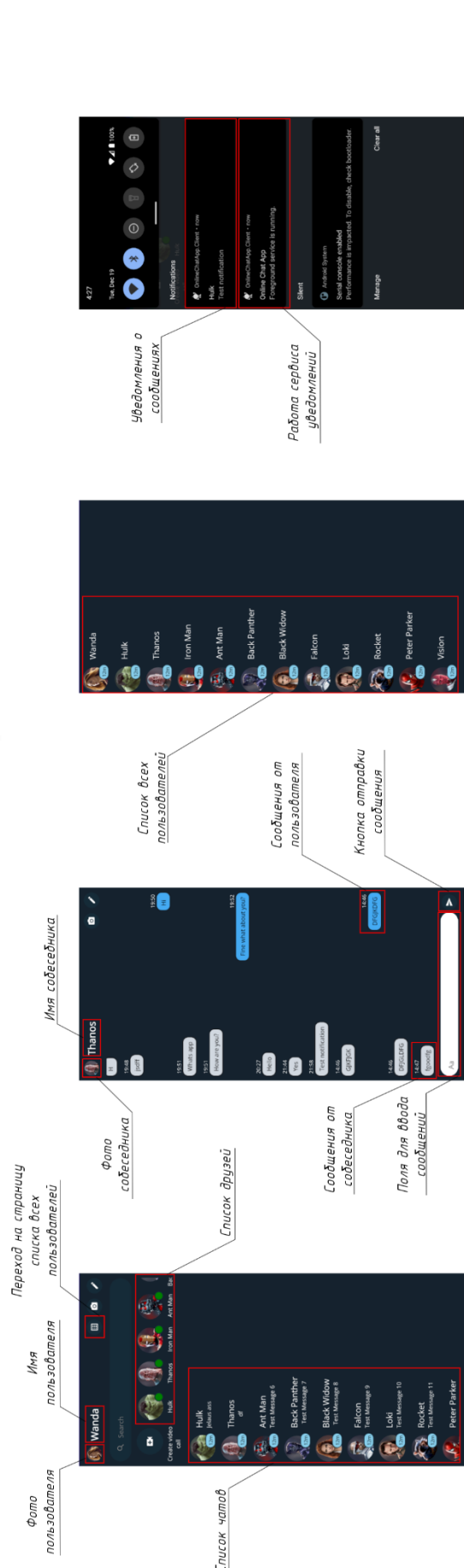


Рисунок 5 – Главная страница

Рисунок 6 – Страница прибитного чата

Рисунок 7 – Страница списка всех пользователей

Рисунок 8 – Уведомления

Рисунок В.5 – Пользовательский интерфейс программного средства

UML Диаграмма Последовательности

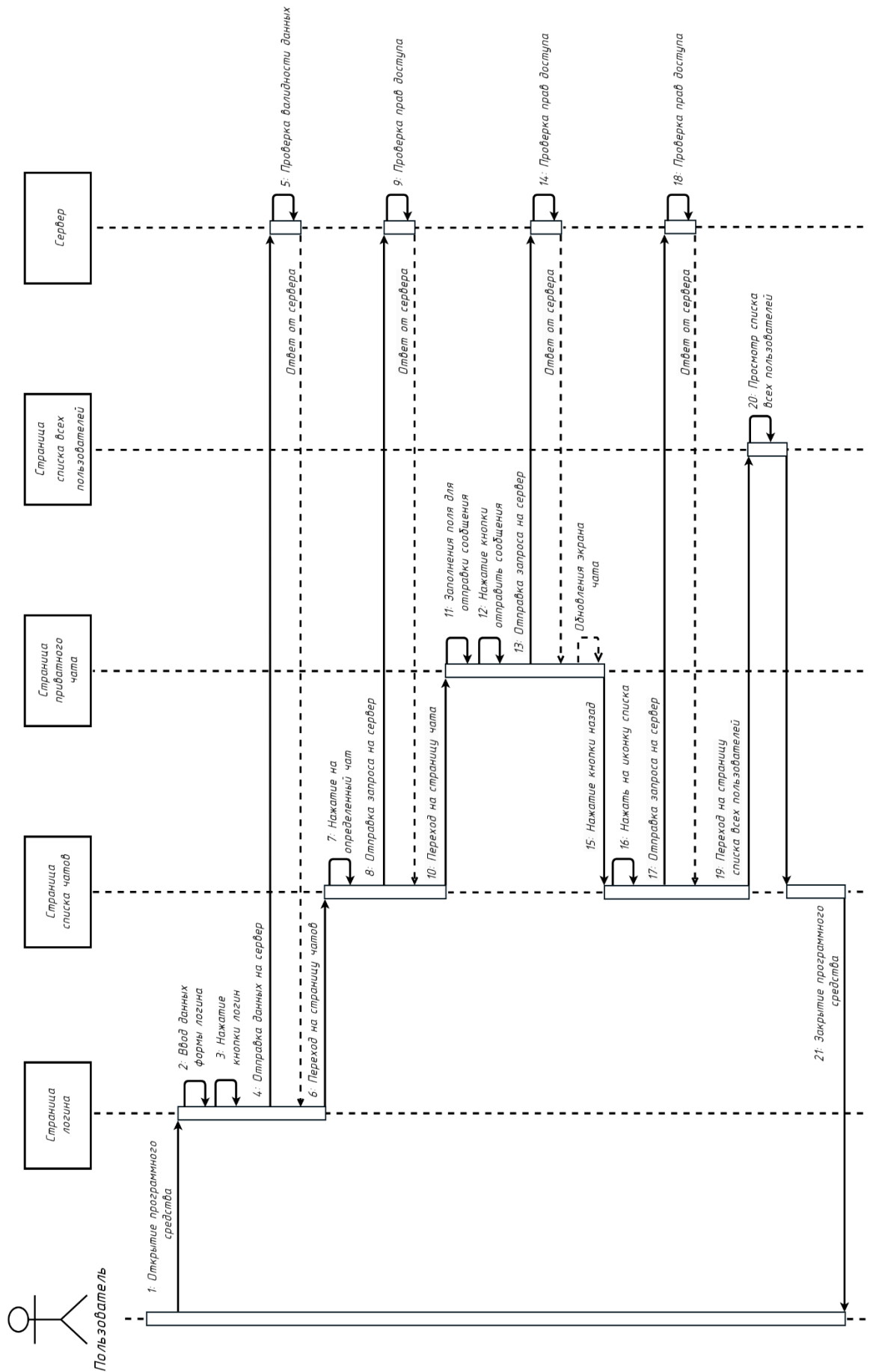


Рисунок В.6 – UML диаграмма последовательности

ПРИЛОЖЕНИЕ Г
(обязательное)
Ведомость дипломной работы