Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ОБМЕНА СООБЩЕНИЯМИ С ФУНКЦИЕЙ СОЗДАНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ЧАТОВ

БГУИР КП 1-40 04 01

Студент: гр. 253505 Павлович В. Ю.

Руководитель: ассистент кафедры информатики Тушинская Е.В.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир развивается стремительно, и новые технологии значительно ускоряют темп нашей жизни. Очень важно не отставать от этого ритма и всегда поддерживать связь с друзьями, знакомыми, семьей, коллегами и руководителями. В этом нам помогают современные технологии. Многие компании создают приложения для общения, позволяющие совершать звонки и обмениваться сообщениями. Эти программные средства, называемые мессенджерами, обеспечивают постоянную связь с нужными людьми, независимо от расстояния.

Мессенджеры помогают нам не только поддерживать контакт, но и облегчают повседневные задачи. Они позволяют оставаться в курсе событий, управлять делами, получать информацию и развлекаться. Эти технологии стали неотъемлемой частью нашей жизни и продолжают развиваться, чтобы соответствовать нашим потребностям и ожиданиям. Благодаря ним мы можем работать удаленно, быстро решать рабочие вопросы и организовывать видеоконференции с коллегами по всему миру.

Кроме того, мессенджеры играют важную роль в экстренных ситуациях, позволяя быстро обмениваться важной информацией и координировать действия. Они также предоставляют платформу для творческого самовыражения и хобби: мы можем делиться своими достижениями, обсуждать интересные темы и находить единомышленников. В социальной сфере мессенджеры помогают нам оставаться на связи с близкими людьми, поддерживать отношения и чувствовать их поддержку, даже если они находятся далеко.

С каждым годом мессенджеры становятся более функциональными и безопасными, интегрируя в себя новые возможности и технологии. Это позволяет нам использовать их в самых разных аспектах жизни, делая её удобнее и продуктивнее.

Кроме того, важно обеспечить доступ пользователей к таким приложениям в любых ситуациях, в чем очень помогает кросс-платформенность. Пользователи могут устанавливать и использовать мессенджер на различных устройствах и операционных системах, что позволяет оставаться на связи, независимо от того, какое устройство у вас под рукой. Пользователи могут выбирать устройство, которое им удобнее использовать в данный момент. Например, на работе можно пользоваться компьютером, а дома – смартфоном или планшетом. Это повышает гибкость и удобство использования.

В данном курсовом проекте будет разработан кросс-платформенный мессенджер, представляющий собой удобное средство для общения между людьми. Пользователи смогут создавать группы или общаться в личных сообщениях, что сможет облегчить его повседневную жизнь.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В данном пункте будут рассмотрены некоторые аналоги разрабатываемого программного средства. В пример будет приведено 3 наиболее популярных мессенджера, каждым из которых ежедневно пользуются миллионы людей по всему миру.

Первый пример, который можно рассмотреть – Telegram, который был разработан Павлом Дуровым в 2013 году и до сих пор активно развивается. Telegram – это веб-приложение для обмена мгновенными сообщениями с акцентом на скорость и безопасность. Это быстрый, простой, безопасный и бесплатный сервис. Он легко синхронизируется на всех устройствах, работает на настольных ПК, планшетах и телефонах. С помощью него можно отправлять неограниченное количество сообщений, фотографий, видео и файлов любого типа. Telegram позволяет пользователям общаться в личных и групповых чатах, а также создавать чаты, в которых может находиться до 200000 пользователей. Также данное приложение позволяет создавать секретные чаты с дополнительным шифрованием и автоудалением данных через заданные промежутки времени.

Второй пример – Viber, разработанный компанией Viber Media в 2010 году. На сегодняшний день это одно из популярных мобильных приложений-мессенджеров. Им активно пользуются для обмена сообщениями, файлами (фото, видео, аудио) и ссылками, а также для звонков и видеозвонков. Программа адаптирована под смартфоны, планшеты и компьютеры и позволяет бесплатно общаться всем зарегистрированным в ней пользователям. Как и Telegram, Viber позволяет общаться как в личных, так и в групповых чатах, однако, в сравнении с телеграмом, у данного приложения существуют некоторые ограничения, например, максимальный размер передаваемого файла в Viber – 10Mb, тогда как Telegram в бесплатной версии позволяет передавать файлы вплоть до 2Gb.

Третий пример – WhatsApp, выпущенный в 2009 году компанией WhatsApp Inc., однако на данный момент приложением владеет компания Meta. WhatsApp – американский бесплатный сервис обмена мгновенными сообщениями и голосовой связи по IP. Он позволяет пользователям отправлять текстовые и голосовые сообщения, совершать голосовые и видеозвонки, обмениваться изображениями, документами, местоположением пользователя и другим контентом. Приложение имеет клиенты для мобильных устройств и персональных компьютеров под управлением различных операционных систем, а также веб-версию для работы через браузер.

После ознакомления с аналогами можно определить требуемый функционал программного средства и поставить задачу по его реализации.

1.2 Постановка задачи

В рамках данного курсового проекта была поставлена следующая задача: реализовать приложение для обмена сообщениями, построенное на клиент-серверной архитектуре.

Проанализировав поставленную задачу, был выделен следующий ряд требований:

– реализовать клиент-серверную архитектуру (отдельное приложение для сервера и отдельное для клиента);

– Реализовать кросс-платформенность, а именно создать клиенты для наиболее популярных операционных систем на мобильных устройствах и персональных компьютерах;

– реализовать возможность регистрации пользователя с помощью логина и пароля;

– реализовать авторизацию пользователя по логину и паролю и предоставление ему только тех данных, к которым он имеет доступ;

– реализовать возможность выхода пользователя из учетной записи;

– реализовать возможность редактирования пользователем его учетной записи;

– реализовать возможность создания многопользовательский чатов и диалогов;

– реализовать возможность добавления или удаления пользователей в существующие многопользовательские чаты;

– реализовать возможность отправки, редактирования и удаления сообщений в чатах с отображением изменений в чате у других пользователей в реальном времени.

Разработав данный набор задач, можно перейти непосредственно к проектированию программного средства.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Общая информация

Для создания данного приложения было решено использовать язык программирования C# и среду разработки Microsoft Visual Studio Community 2022. Данный продукт представляет собой полнофункциональную, расширяемую, бесплатную среду разработки со встроенным редактором кода, компилятором, отладчиком и инструментами Git для создания различных приложений на выбранном языке программирования. Также данная платформа включает в себя функции автодополнения кода, анализа ошибок и анализа производительности. Так же Visual Studio полностью интегрирована с платформой .NET, что обеспечивает широкие возможности для разработки на C#, что делает ее идеальным инструментом для выбранных целей, удобство работы на протяжении всего процесса разработки приложения.

C# – это общецелевой объектно-ориентированный язык программирования, выпущенный компанией Microsoft в 2002 году. Данный язык полностью ориентирован на объекты, что делает его гибким и удобным для разработки и поддержки сложных приложений. C# был создан специально для работы с платформой .NET, что вносит ряд своих преимуществ. Основной средой исполнения данного языка является общеязыковая среда исполнения CLR (Common Language Runtime), которая поддерживает несколько языков, что позволяет разрабатывать отдельные модули приложения на различных языках программирования, таких как C#, VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. .NET является переносимой платформой, что обеспечивает кроссплатформенность. Например, последняя версия платформы на данный момент –.NET 8 поддерживается на большинстве современных ОС: Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ — Windows, MacOS, Linux, Android, iOS, Tizen. Также язык C# обеспечивает безопасную работу с памятью – он имеет встроенный сборщик мусора, который автоматически управляет памятью и предотвращает ее утечки.

Данная платформа и язык программирования активно поддерживаются компанией Microsoft – каждый год выходят новая версия языка C# и платформы .NET, расширяющие функциональные возможности данных технологий.

Для разработки API серверного приложения использовался фреймворк ASP.NET Core. Данный фреймворк представляет собой кросс-платформенную технологию с открытым исходным кодом 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#.

Для разработки пользовательского интерфейса клиентского приложения использовался фреймворк .NET MAUI. NET MAUI (Multi-platform App UI) – это кросс-платформенный фреймворк для создания мобильных и настольных приложений с использованием C# и XAML. С помощью .NET MAUI можно разрабатывать приложения, которые могут работать на Android, iOS, iPadOS, macOS и Windows, используя единую общую кодовую базу.

Для работы с базой данный использовался Entity Framework Core. Entity Framework Core (EF Core) – это легковесная, расширяемая, открытая и кросс-платформенная версия Entity Framework, популярного Object-Relational Mapping (ORM) фреймворка от Microsoft для работы с базами данных. EF Core представляет собой полностью переписанную версию Entity Framework, которая позволяет разработчикам работать с базами данных с использованием .NET объектов.

В качестве системы управления базами данных (СУБД) была выбрана MySQL. Она представляет собой систему управления реляционными базами данных. На сегодняшний день это одна из самых популярных систем управления базами данных. Изначальным разработчиком данной СУБД была шведская компания MySQL AB. В 1995 году она выпустила первый релиз MySQL. В 2008 году компания MySQL AB была куплена компанией Sun Microsystems, а в 2010 году уже компания Oracle поглотила Sun и тем самым приобрела права на торговую марку MySQL. Поэтому MySQL на сегодняшний день развивается компанией Oracle. MySQL обладает кроссплатформенностью, имеются дистрибутивы под самые различные ОС, в том числе наиболее популярные версии Linux, Windows, MacOS.

2.2 Разработка функциональности программного средства

После анализа требований к проектируемому программному средству были выделены следующие возможности пользователя, представленный на диаграмме сценариев на рисунке 2.1.

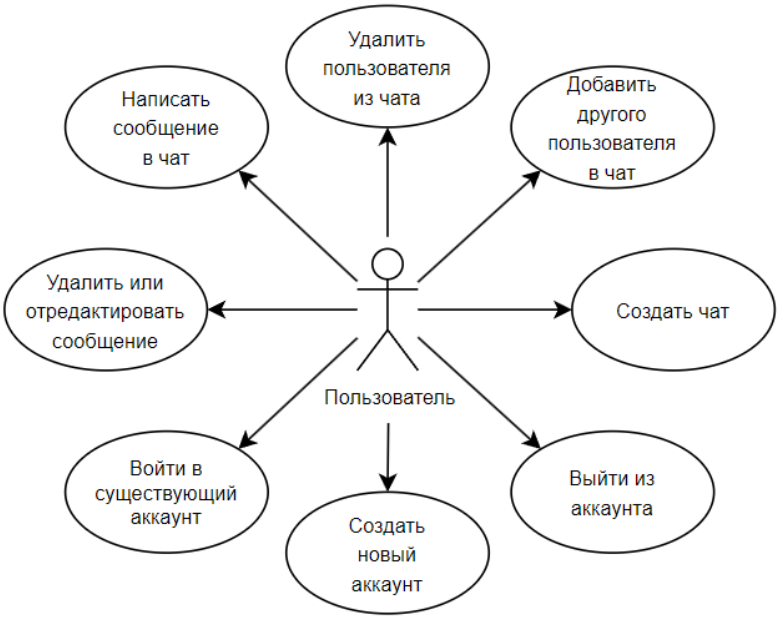

Рисунок 2.1 - Use-case диаграмма приложения

На основе представленной диаграммы можно перейти непосредственно к проектированию архитектуры приложения.

2.3 Архитектура программного средства

В качестве архитектуры для клиентского и серверного приложений была выбрана чистая архитектура. Она предлагает разбиение приложения на независимые функциональные компоненты, которые взаимодействуют друг с другом определенным способом. Между ними передаются только те ресурсы, которые необходимы для выполнения поставленной задачи.

При разработке приложения использовались следующие методологии разработки:

– Посредник – это поведенческий паттерн проектирования, который позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник;

– Репозиторий – это архитектурный паттерн, который предлагает абстракцию слоя доступа к данным. Его цель – скрыть детали реализации базы данных или другого хранилища данных от остальной части приложения;

– Unit Of Work – это архитектурный паттерн, который помогает упростить работу с различными репозиториями и обеспечивает единое управление изменениями в базе данных в рамках одной логической транзакции;

– Внедрение зависимостей – процесс предоставления внешней зависимости программному компоненту. Является специфичной формой

«инверсии управления» (Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с принципом единственной ответсвенности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму;

– Data Transfer Object (DTO) – шаблон проектирования, используемый для определения формата, в котором данные передаются между клиентским и серверным приложениями.

В серверном приложении можно выделить следующие слои:

– Доменный слой – определяет модели, используемы в приложении и базе данных, а также абстракции репозиториев для управления данными моделями;

– Слой инфраструктуры – обеспечивает доступ к данным, реализует паттерны репозиторий и Unit of Work, содержит реализацию абстракции репозитория и классов для работы непосредственно с базой данных. Вся работа с базой данных происходит в репозиториях, которые объединены классов Unit of Work, организующим всю работу с базой данных;

– Слой приложения – реализует паттерн посредник. Данный слой предоставляет все требуемые для работы приложения сценарии использования, что позволяет ограничить доступ к базе данных для верхних слоев архитектуры приложения. Данный слой обеспечивает независимость логики работы приложения от реализации слоя представления;

– Слой представления – содержит реализованный с использование фреймворка ASP.NET Core веб-интерфейс приложения. Вся работа с нижними слоями производится только через класс-посредник из слоя приложения. Также реализует систему авторизации и аутентификации пользователей.

В клиентском приложении можно выделить следующие слои:

– Доменный слой – определяет модели, используемы в приложении;

– Слой доступа к данным – содержит классы, реализующие паттерны репозиторий и Unit of Work, обеспечивает доступ к данным, находящимся на сервере посредством обращений к программному веб-интерфейсу серверного приложения;

– Слой представления – содержит реализованный с использованием .NET MAUI графический пользовательский интерфейс приложения, позволяющий пользователю получить полный доступ ко всем возможностям данного программного средства.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка серверного приложения

Так как разрабатываемый программный продукт будет состоять из двух приложений, а именно клиентского, с которым непосредственно взаимодействуют пользователи, и серверного, с которым взаимодействуют клиентские приложения, необходимо отдельно рассмотреть разработку каждого из них в том порядке, в котором они создавались. В данном подразделе будет рассмотрено серверное приложение.

3.1.1 Разработка слоя представления

Данный слой реализован с использованием фреймворка ASP.NET Core. На данном слое содержатся классы для контроллеров веб-интерфейса приложения, определения моделей для обмена данными с клиентом и класс для преобразования доменный моделей в DTO модели, класс для работы с узлом подключения к приложению для обмена данными между различными клиентами в реальном времени а также точка входа в приложения. Рассмотрим отдельно каждую из этих групп классов.

Классы контроллеров веб-интерфейса приложения предоставляют публичное API (программный интерфейс приложения) для организации доступа к данным приложения клиентам. Вся внутренняя реализация методов данных классов взаимодействует только со слоем приложения, к каждому из них через внедрение зависимостей добавлен объект класса посредника. Рассмотрим отдельно каждый из этих контроллеров:

– AuthController – данный класс содержит в себе только два метода: register() и login(), которые соответственно дают доступ к регистрации и входу в аккаунт пользователя. При обращении к методу регистрации в теле запроса передается информация о пользователе, а именно имя пользователя, логин и пароль. Данные запроса валидируются и в случае корректности пользователь с данными параметрами добавляется в базу данных. Метод login() проверяет корректность полученных от пользователя данных и производит поиск такого пользователя в базе данных. В случае успеха каждый из этих методов возвращает такую информацию о пользователе, как имя пользователя и его уникальный идентификатор, а также токен авторизации, необходимый для доступа к методам остальных контроллеров приложения;

– UserController – данный класс содержит в себе методы для изменения информации о пользователе, а именно изменения имени или пароля пользователя. Также в данном контроллере содержится метод для получения чатов, в которых состоит пользователь. Для обращения к методам данного контроллера необходим токен авторизации, который проверяется при каждом

обращении и запрещает несанкционированный доступ к данным (например, имея токен одного пользователя, нельзя изменить имя или пароль другого пользователя, а также получить доступ к списку его чатов). В случае отсутствия такого токена или при несоответствии токена пользователю, методы данного контроллера вернут ошибку доступа;

– ChatController – данный класс содержит в себе все методы для работы с чатами, такие как создание или редактирование чатов, добавление пользователей в чаты и удаление пользователей из них, отправка, удаление и редактирование сообщений а также получение списка пользователей или сообщений конкретного чата. Для доступа к данному контроллеру также необходим токен авторизации, при каждом запросе по содержимому токена проверяется, имеет ли пользователь доступ к данному чату и возвращается соответствующий результат.

Далее рассмотрим класс преобразования доменных моделей в модели для предоставления данных пользователю. Данный процесс реализован с помощь библиотеки AutoMapper и представлен классов AppMappingProfile, в котором определены все возможные профили преобразования доменных моделей в модели для обмена данными. Данная библиотека предоставляет интерфейс IMapper и его реализацию, которую можно внедрить через внедрение зависимостей. Данный интерфейс содержит шаблонный метод Map, производящий преобразование объекта одного класса в объект другого класса в соответствии с выбранным профилем преобразования, определенном в классе AppMappingProfile.

Для реализации узла подключения к приложению использовалась библиотека MediatR, дающая возможность серверу отправлять данные клиентам без их непосредственного запроса. Данная возможность представлена классом MessengerHub, являющимся наследником класса Hub из библиотеки MediatR. Данный класс содержит методы для подключения к узлу или отключения от него и методы для рассылки данных пользователям. Для подключения к узлу необходим токен авторизации. В качестве примера работы данного класса можно привести следующий алгоритм: пользователь отправляет сообщение в чат, после чего отправляет соответствующий запрос в узел подключения. Далее сервер обрабатывает этот запрос и отправляет данное сообщение всем подключенным к нему пользователям, состоящим в этом чате. Данная технология позволяет в реальном времени отображать новые сообщения в чате, что критически важно в данном программном средстве. Также через этот узел передается информация об редактировании или удалении сообщений, создании чатов и добавлении или удалении в них пользователей.

В точке входа в приложении происходит подключение всех слоев приложения друг другу через внедрение зависимостей, подключение внешних зависимостей, подключение приложения к базе данный и непосредственно запуск веб-сервера.

3.1.2 Разработка слоя приложения

Слой приложения реализован с использованием библиотеки MediatR, которая добавляет интерфейс IMediatr и его реализацию, которую можно внедрить через внедрение зависимостей. Данный класс содержит метод Send, отправляющий некоторый запрос к слою приложения и возвращающий результат работы соответствующего ему обработчика. На данном слое определен список запросов, которые представляют собой класс запроса, наследующийся от шаблонного интерфейса IRequset, конструктор которого содержит требуемые параметры, а также класс-обработчик данного запроса, наследующийся от шаблонного интерфейса IRequestHandler, содержащий метод Handle, который вызывается при поступлении определенного запроса через объект класса, реализующего интерфейс IMediatr. Данный слой реализован с использованием принципа CQRS (Command and Query Responsibility Segregation), который разделяет запросы на команды, которые изменяют содержимое базы данных, и очереди, которые получают содержимое базы данных. Каждый метод обработки запроса работает с объектом, реализующим интерфейс IUnitOfWork, имеющим непосредственный доступ к базе данных. Данный слой является посредником между инфраструктурным слоем и слоем представления, задающим определенный набор действий, необходимых для корректной работы приложения, что облегчает разработку слоя представления и позволяет в случае необходимости полностью переделать слой представления с использованием других технологий с меньшими трудозатратами.

3.1.3 Разработка слоя доступа к данным

Слой инфраструктуры обеспечивает доступ приложения доступ к данным посредством реализации паттерна репозиторий, а также облегчает работу с репозиториями посредством реализации паттерна UnitOfWork. Рассмотрим классы, находящиеся на данном слое.

AppDbContext – класс, наследующийся от базового для Entity Framework Core класса DbContext. В данном классе задается структура таблиц базы данных и проверяется существование базы данных при подключении к ней. В случае отсутствия необходимой базы данных, создается пустая база данных с требуемыми параметрами таблиц.

EfRepository<T> – шаблонный класс, реализующий интерфейс IRepository с доменного слоя. Данный класс имеет доступ к таблице базы данных, содержащей объекты типа T и содержит следующие методы:

– GetByIdAsync() – метод, возвращающий объект из базы данных с требуемым идентификатором. В случае передачи соответствующих параметров также может вместе с самим объектом вернуть объекты из других таблиц,

связанных с ним внешними ключами;

– ListAllAsync() – возвращает все объекты из таблицы базы данных, связанной с данным конкретным репозиторием;

– ListAsync() – возвращает все объекты из таблицы базы данных, соответствующие переданному фильтру. В случае передачи соответствующих параметров также может вместе с каждым из них вернуть объекты из других таблиц, связанных с ним внешними ключами;

– AddAsync() – добавляет переданный объект в базу данных;

– UpdateAsync() – изменяет переданный объект в базе данных;

– DeleteAsync() – удаляет переданный объект из базы данных;

– FirstOrDefaultAsync() – возвращает первый объект из базы данных, соответствующий переданному фильтру. В случае отсутствия такого объекта в базе данный, возвращает null.

EfUnitOfWork – класс, реализующий интерфейс IUnitOfWork из доменной модели. Содержит в себе объекты класса EfRepository для каждой модели, а также методы CreateDataBaseAsync() и DeleteDataBaseAsync(), которые создают и удаляют базу данных соответственно, а также метод SaveAllAsync(), который сохраняет внесенные в базу данных изменения.

3.1.4 Разработка доменного слоя

Доменный слой содержит определение используемый для хранения в базе данных моделей, а также интерфейсы, организующие работу с данными моделями.

Все классы моделей наследуются от базового класса Entity, содержащего только одно поле – Id, необходимое для корректного их хранения в базе данных и используемого в качестве первичного ключа в базе данных. От него наследуются классы User, Message и Chat, содержащие соответствующую информацию об объектах. Также на данном слое определены интерфейсы IRepository<T> и IUnitOfWork, которые содержат методы для организации работы с моделями. Методы данных интерфейсов были рассмотрены в предыдущем пункте на примере их конкретных реализаций.

3.2 Разработка клиентского приложения

После разработки серверного приложения и реализации программного интерфейса для работы с ним, рассмотрим разработку клиентского приложения. Для взаимодействия этих двух приложений используется технология REST API, позволяющая разделить HTTP запросы на различные группы, в зависимости от их назначения. В процессе взаимодействия пользователя с клиентским приложением, оно будет посылать данные запросы на сервер для взаимодействия

с данными, находящимися на сервере, и получения данных, актуальных для всех пользователей, в реальном времени.

3.2.1 Разработка слоя представления

Слой представления клиентского приложения реализован с использованием фреймворка .NET MAUI. Данная технология позволяет с использованием языка разметки XAML создавать различные страницы приложения, которые могут отображаться пользователю при совершении им определенных действий. Также .NET MAUI позволяет с использованием языка C# описывать логику работы каждой страницы при обработке различных действий со стороны пользователя. Также на данном слое располагается точка входа в приложение, в которой подключаются необходимые внешние зависимости и происходит внедрение требуемых сущностям приложения зависимостей. Рассмотрим каждую из представленных в приложении страниц. Данный слой не взаимодействует напрямую с доменным слоем и не занимается отправкой запросов на сервер, а использует классы, реализованные на следующем слое, которые организуют правильную работу с данными возможностями. Такое разделение ответственности между слоями позволяет совершать меньше ошибок, так как все необходимые методы уже будут реализованы.

LogInPage – данная страница отображается при запуске приложения. На ней представлены 2 поля для ввода текста: поле ввода логина и пароля, а также кнопка для входа в приложение и кнопка для регистрации. При нажатии на кнопку входа в приложение данные из полей ввода отправляются на сервер, и в случае их корректности происходит переход на следующую страницу приложения. Если данные были введены некорректно либо при отсутствии подключения к серверу появляется соответствующее всплывающее окно. Нажатие на кнопку регистрации перенаправляет пользователя на страницу регистрации. На рисунке 3.1 представлен внешний вид данной страницы на мобильном устройстве на платформе Android. Все следующие рисунки данного раздела также будут представлены для платформы Android.

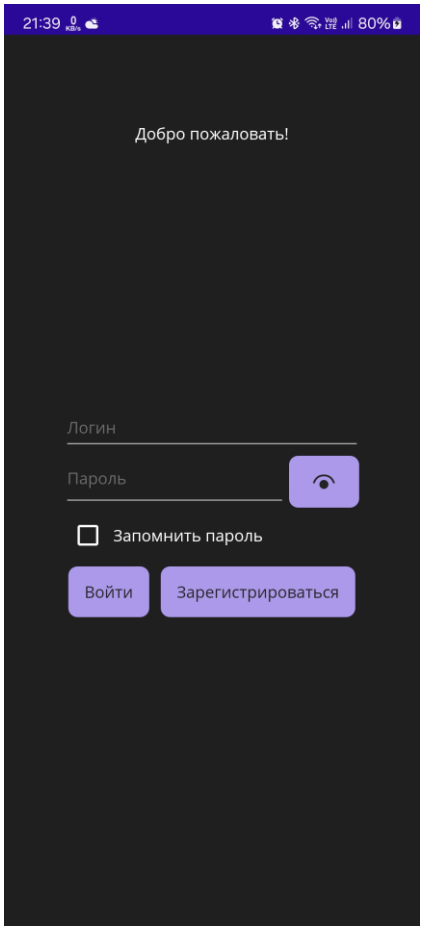

Рисунок 3.1 – Экран входа в приложение для платформы Android

RegisterPage – данная страница отображается при нажатии на кнопку регистрации на странице входа. Она содержит 4 текстовых поля: имя пользователя, логин, пароль и повтор пароля, а также кнопку для регистрации и кнопку для возврата на страницу входа. При нажатии на кнопку регистрации происходит проверка введенных пользователем данный, после чего происходит отправка запроса на регистрацию пользователя на сервер. В случае некорректности введенных данный, отсутствия подключения к серверу или при наличии существующего пользователя с введенным логином появляется соответствующее уведомление. В случае корректности данных осуществляется переход на следующую страницу. Внешний вид данной страницы представлен на рисунке 3.2.

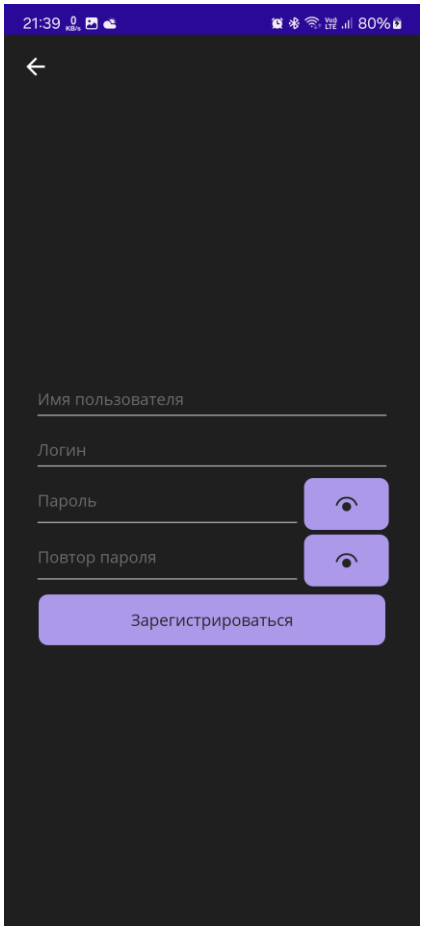

Рисунок 3.2 – Экран регистрации пользователя для платформы Android

ChatsPage – на данной странице представлен список чатов пользователя. В данном списке отображается название чата, а также последнее отправленное в него сообщение вместе с именем его отправителя и датой отправки. В диалогах не отображается имя отправителя, если последнее сообщение было отправлено пользователем, находящимся на этой странице. При нажатии на конкретный чат происходит перенаправление пользователя на страницу этого чата. Также на этой странице находятся кнопки для создания нового чата и отправки личного сообщения любому пользователю и кнопка для перехода на другие страницы, а именно на страницу пользователя. При нажатии на эти кнопки появляются соответствующие всплывающие окна, позволяющие выполнить данную операцию. Переход на страницу чатов осуществляется после входа пользователя в приложение через страницу входа или регистрации. Внешний вид данной страницы представлен на рисунке 3.3.

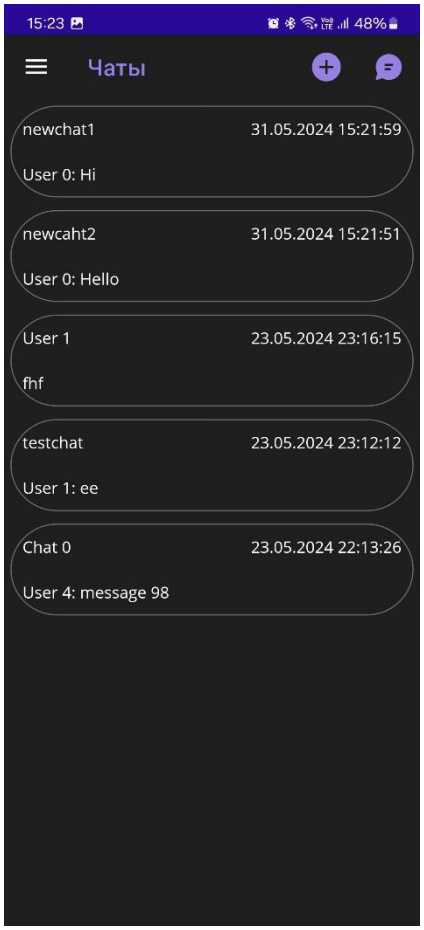

Рисунок 3.3 – Экран списка чатов для платформы Android

ProfilePage – на данной странице представлено текстовое поле с именем пользователя, а также кнопки для смены имени пользователя, пароля, перехода на другую страницу и выхода из аккаунта. Нажатие на кнопку смены имени пользователя или пароля вызывает соответствующее кнопке всплывающее окно. Нажатие кнопки выхода из аккаунта перенаправляет пользователя на страницу входа в приложение. Нажатие на кнопку смены страницы вызывает всплывающее окно, в котором можно осуществить переход на страницу чатов данного пользователя. Для попадания на данную страницу пользователю на странице чатов необходимо нажать на кнопку списка страниц и выбрать там страницу профиля. При попытке смены имени пользователя или пароля происходит проверка корректности введенных данных, после чего появляется всплывающее окно, сообщающее об успешности проведения данного действия. В случае успеха внесенные изменения сохраняются на сервере. Внешний вид данной страницы представлен на рисунке 3.4.

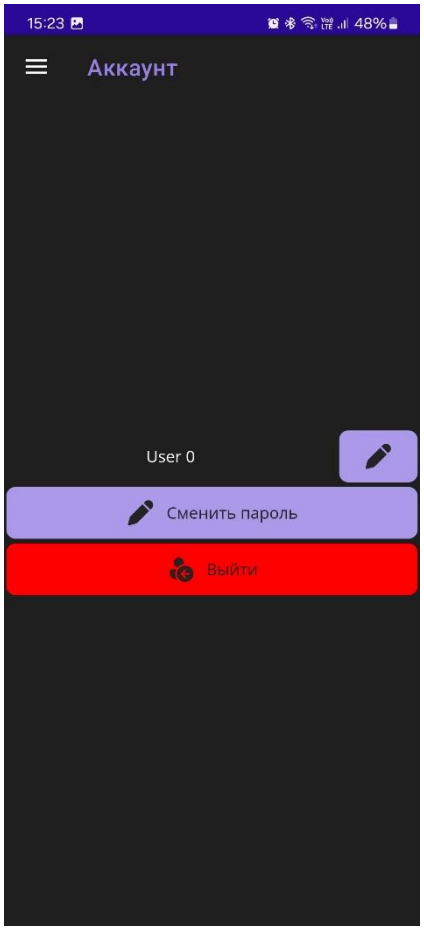

Рисунок 3.4 – Экран страницы профиля для платформы Android

CurrentChatPage – на этой странице находится список всех сообщений конкретного чата, текстовое поле для ввода нового сообщения и кнопка для его оправки, текстовое поле с названием текущего чата и кнопка для возврата на предыдущую страницу. Для попадания на эту страницу пользователю необходимо нажать на соответствующий чат на странице чатов. После ввода текста в текстовое поле и нажатия на кнопку отправки данное сообщение отправляется на сервер, после чего информация о добавлении сообщения отправляется всем пользователям, состоящим в данном чате. В списке сообщений в каждом элементе списка содержится текст сообщения, имя отправителя и время отправки. Также при отображении списка сообщения, отправленные пользователем, который непосредственно находится на данной странице, выровнены по правому краю, а чужие сообщения – по правому. При нажатии на свое сообщение появляется всплывающее окно с кнопками для редактирования или удаления сообщения. При нажатии на название чата происходит перенаправление на страницу информации о чате, но данная возможность справедлива только в случае, если чат не является диалогом. Внешний вид страницы представлен на рисунке 3.5.

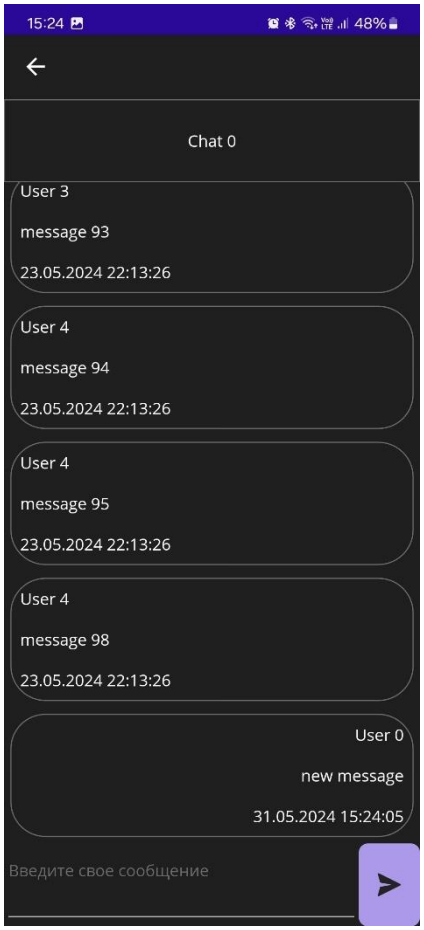

Рисунок 3.5 – Экран страницы чата для платформы Android

ChatInfoPage – данная страница предоставляет информацию о текущем чате. Она состоит из текстового поля с названием чата, кнопок добавления пользователя и изменения имени чата, кнопка перехода на предыдущую страницу а также списка участников данного чата. Напротив каждого из участников чата находится кнопка удаления пользователя из чата. Внешний вид данной страницы представлен на рисунке 3.6.

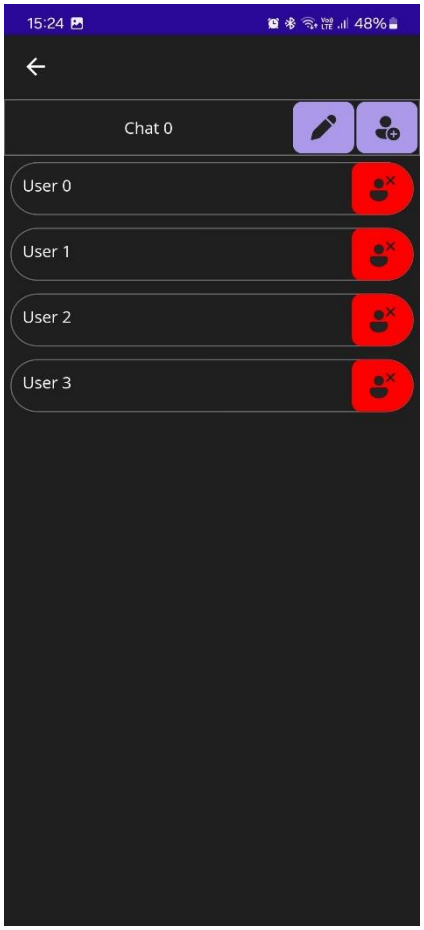

Рисунок 3.6 – Экран страницы информации о чате для платформы Android

3.2.2 Разработка слоя доступа к данным

Данный слой содержит в себе классы для организации доступа к данным. Он состоит из классов-репозиториев для каждой из используемых в приложении моделей, класс UnitOfWork, объединяющий в себе все репозитории а также интерфейс IServerService и его реализацию ServerService, организующие работу с серверным приложением. Внутри методов репозиториев для работы с данными сервера используется объект класса ServerService.

Класс ServcerService организует отправку HTTP запросов к серверному приложению и обработку ответов от сервера, в том числе и обработку ошибок. Также с использованием библиотеки SignalR в данном классе осуществляется подключение к узлу подключения серверного приложения для обмена данными между пользователями в реальном времени. Подключение к данному узлу происходит после успешного входа пользователя в приложение. Получение данных от сервера в реальном времени организовано с помощью механизма событий. При получении данных от узла подключения данный класс вызывает соответствующее вызванному методу узла событие, которое обрабатывается в

соответствующих репозиториях. Также в данном классе при вызове методов авторизации или регистрации перед отправкой запроса на сервер происходит хеширование пароля с использованием алгоритма SHA256.

3.2.3 Разработка доменного слоя

Доменный слой клиентского приложения содержит в себе определение моделей, используемый в приложении. В нем определены такие классы, как User, Message, Chat и базовый для них чат Entity, необходимый для корректной организации шаблонных классов, использующих данные модели. Каждая из моделей содержит поля для необходимых ей данных и свойства, организующие доступ к этим данным для классов, работающих непосредственно с моделями.

3.3 Разработка контекста базы данных

В качестве системы управления базами данных была выбрана MySQL. Выбор именно реляционной базы данных был важен по причине того, что каждая из моделей имеет какие-либо зависимости в отношении других моделей. Итоговая схема таблиц базы данных представлена на рисунке 3.7

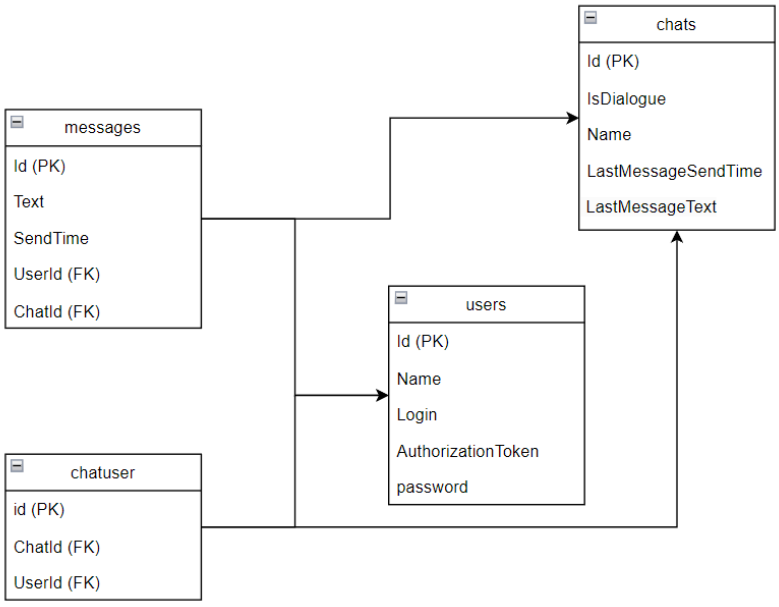

Рисунок 3.7 – Схема базы данных

После завершения этапа разработки программного средства можно перейти к проверке работоспособности полученного приложения.

4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРИЛОЖЕНИЯ

Осуществлялось функциональное тестирование; отчёт о проведённом тестировании представлен в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

№	Тестируемая функция	Ожидаемый результат	Полученный результат
1	Нажатие кнопки входа в приложение с корректно введенными данными	Пользователь перенаправляется на страницу со списком его чатов	Соответствует заданному результату
2	Нажатие кнопки входа в приложение с некорректными данными	Появляется всплывающее окно с сообщением о некорректности введенных данных. Пользователь остается на странице входа	Соответствует заданному результату
3	Нажатие на чат на странице списка чатов	Пользователь перенаправляется на страницу выбранного чата	Соответствует заданному результату
4	Нажатие кнопки отправки сообщения в чат с введенным сообщением в необходимое текстовое поле на странице чата	Сообщение появляется в списке сообщений чата у отправителя и других участников чата	Соответствует заданному результату
5	Нажатие кнопки удаления сообщения на странице чата	Сообщение удаляется из списка сообщений у всех участников чата	Соответствует заданному результату
6	Нажатие кнопки добавления пользователя в чат на странице информации о чате и выбор пользователя для добавления	Выбранный пользователь появляется в списке участников чата у всех остальных участников. Данный чат появляется в списке чатов добавленного пользователя	Соответствует заданному результату

Продолжение таблицы 4.1

7	Нажатие кнопки регистрации на странице регистрации с корректно введенными данными	В базе данных на сервере создается запись о новом пользователе. Пользователь переходит на пустую страницу списка чатов	Соответствует заданному результату
8	Нажатие кнопки регистрации на странице регистрации с некорректными данными	Появляется всплывающее окно, в котором описана причина ошибки	Соответствует заданному результату
9	Нажатие кнопки удаления пользователя из чата на странице информации о чате	Выбранный пользователь удаляется из списка участников чата. Данный чат удаляется из списка чатов выбранного пользователя	Соответствует заданному результату
10	Нажатие кнопки выхода из аккаунта на странице профиля	Пользователь перенаправляется на страницу входа в приложение без возможности возврата на предыдущие страницы	Соответствует заданному результату

В ходе тестирования не было выявлено ошибок, связанных с некорректным поведением приложения в различных сценариях использования. Программное средство продемонстрировало высокую стабильность работы и соответствие ожидаемым результатам в процессе использования.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Начало работы

Для начала работы пользователю необходимо установить приложение на устройство и запустить его. Перед пользователем появится экран входа в приложение, на котором он сможет ввести логин и пароль для входа в аккаунт. В случае отсутствия у пользователя аккаунта, он должен нажать на кнопку регистрации и создать новый аккаунт. После любого из этих действий пользователь попадет на страницу чатов.

На странице чатов перед пользователем будет представлен список его чатов и личных диалогов с другими пользователями. Для входа в чат необходимо нажать на него в данном списке. Далее пользователь попадет на страницу чата. Если пользователю необходимо создать новый чат, ему необходимо нажать на иконку с плюсом в верхней части экрана и ввести название нового чата, после чего он попадет на его страницу. Если пользователю необходимо написать личное сообщение какому-либо другому пользователю, то он должен нажать на иконку сообщения в верхней части экрана и ввести в строке поиска имя пользователя, которому он хочет написать, после чего нажать на данного пользователя в списке и написать ему сообщения.

На странице чата для отправки сообщения пользователю необходимо ввести текст сообщения в строку ввода и нажать на кнопку и иконкой отправки сообщения. Если ему понадобится изменить или удалить какое-либо из своих сообщений, он должен нажать на него в списке сообщений чата и во всплывающем окне нажать на соответствующую кнопку.

Для добавления пользователей в чат необходимо нажатием на название чата на странице чата перейти на страницу информации чата. Далее необходимо нажать на иконку добавления пользователя в чат и в строке поиска ввести имя необходимого пользователя, после чего нажать на соответствующую кнопку напротив информации о нем. Для удаления пользователя из чата необходимо нажать на кнопку удаления напротив имени пользователя в списке участников чата.

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом было разработано кросс-платформенное приложение для обмена сообщениями, реализующее клиент-серверную архитектуру. Оно полностью удовлетворяет всем поставленным задачам.

Реализация проекта состояла из нескольких этапов: разработка требований и проектирование, реализация поставленных задач и тестирование работоспособности приложения.

На первом этапе были разработаны требования к проекту, а также были выбраны необходимые для разработки технологии и спроектирована архитектура приложения. На втором этапе с использованием выбранных технологий производилась реализация приложения с учетом поставленных требований. На третьем этапе производились тестирование приложения и исправление допущенных на предыдущем этапе недочетов.

Проект выполнен в соответствии с принципами SOLID и правилами чистой архитектуры, что обеспечивает легкость в расширении функциональности. Например, в будущем в проект могут быть добавлены такие функции, как отправка файлов и изображений в чатах и диалогах. Также благодаря реализации клиент-серверной архитектуры имеется возможность в случае необходимости реализовать веб-версию данного приложения. Однако уже на данном этапе приложение обеспечивает возможность обмениваться сообщениями между пользователями в личных переписках или групповых чатах без каких-либо ограничений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Мартин, Р. Чистая архитектура: искусство разработки программного обеспечения / Р. Мартин. – Санкт-Петербург : Питер, 2024. – 352 с.

[2] Гамма, Э. Паттерны объектно-ориентированного проектирования / Э. Гамма [и др.]. – Санкт-Петербург : Питер, 2021. – 448 с.

[3] Прайс, М. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Прайс. – 6-е изд. – Санкт-Петербург : Питер, 2024. – 848 с.

[4] Лок, Э. ASP.NET CORE в действии / Э. Лок. – 2-е изд. – Москва : ДМК-Пресс, 2021. – 906 с.

[5] Liberty, J. .NET MAUI for C# Developers: Build cross-platform mobile and desktop applications / J. Liberty, R. Juarez. – Birmingham, UK : Packt Publishing, 2023. – 296 с.

[6] Microsoft Learn [Электронный ресурс]. - Режим доступа : https://learn.microsoft.com/.

[7] Metanit [Электронный ресурс]. - Режим доступа : https://metanit.com/.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы

Листинг А.1 – Точка входа в серверное приложение

```
using Server.Application.Temp;
using Microsoft.EntityFrameworkCore;
using Server.Infrastructure.Persistence.Data;
using System.Reflection;
using Microsoft.OpenApi.Models;
using Server.API.Mapping;
using Server.API.Hubs;

string settingsStream = "Server.API.appsettings.json";

var builder = WebApplication.CreateBuilder(args);

var a = Assembly.GetExecutingAssembly();
using var stream = a.GetManifestResourceStream(settingsStream);
builder.Configuration.AddJsonStream(stream!);

builder.Services.AddSignalR();
// Add services to the container.
builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(opt =>
{
    opt.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        BearerFormat = "JWT",
        Scheme = "Bearer"
    });

    opt.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Name = "Bearer",
                In = ParameterLocation.Header,
                Reference = new OpenApiReference
                {
                    Id = "Bearer",
                    Type = ReferenceType.SecurityScheme
```

```
                    }
                },
                new List<string>()
            }
        });
    });

    var connStr =
builder.Configuration.GetConnectionString("MySQLConnection");
    ServerVersion vesrion = ServerVersion.AutoDetect(connStr);

    builder.Services.AddInfrastructure()
                    .AddDbContext<AppDbContext>(opt =>
                        opt.UseMySql(connStr, new MySqlServerVersion(new
Version(8, 0, 36)),
                                             opt => opt.EnableRetryOnFailure()),
                        ServiceLifetime.Scoped)
                    .AddApplication();

    builder.Services.AddAutoMapper(typeof(AppMappingProfile));

    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    app.UseHttpsRedirection();

    app.UseAuthentication();
    app.UseAuthorization();

    app.MapControllers();

    app.MapHub<MessenderHub>("/messenger");

    app.Run();
```

Листинг А.2 – Точка входа в клиентское приложение

```
using Client.Pages;
using Client.Persistence;
using CommunityToolkit.Maui;
using Microsoft.Extensions.Logging;

namespace Client;

public static class MauiProgram
{
```

```
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .UseMauiCommunityToolkit()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf",
"OpenSansSemibold");
                    fonts.AddFont("FluentSystemIcons-Filled.ttf",
"FluentIcons");
                });

            builder.Services.AddPersistence()
                            .AddSingleton<ChatsPage>()
                            .AddTransient<ProfilePage>();

    #if DEBUG
                builder.Logging.AddDebug();
    #endif

            return builder.Build();
        }
    }
```

Листинг А.3 – Реализация контроллера на примере класса UserController

```
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.AspNetCore.Authorization;
    using Server.API.DTO;
    using AutoMapper;
    using System.Security.Claims;

    namespace Server.API.Controllers;

    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class UserController : Controller
    {
        private readonly IMediator _mediator;
        private readonly IMapper _mapper;
        private readonly ILogger _logger;

        public UserController(IMediator mediator, IMapper mapper,
ILogger<UserController> logger)
        {
            _mediator = mediator;
            _mapper = mapper;
            _logger = logger;
```

```
        }

        [HttpPut("updateName")]
        public async Task<IActionResult> UpdateUser(UserDTO request)
        {
            _logger.LogInformation($"Processing route: {Request.Path.Value}");

            var userId =
Convert.ToInt32(HttpContext.User.FindFirstValue("Id"));
            if (userId != request.Id)
            {
                _logger.LogError($"{Request.Path.Value}: 403 Forbidden");
                return Forbid();
            }

            var user = await _mediator.Send(new
GetUserByIdRequest(request.Id));
            if (user is null)
            {
                _logger.LogError($"{Request.Path.Value}: 404 Not Found");
                return NotFound();
            }
            user.Name = request.Name;
            await _mediator.Send(new UpdateUserRequest(user));
            var userDto = _mapper.Map<UserDTO>(user);
            _logger.LogInformation($"{Request.Path.Value}: 200 OK");
            return Ok(userDto);
        }

        [HttpPut("updatePassword")]
        public async Task<IActionResult> UpdateUserPassword(ChangePasswordDTO
request)
        {
            _logger.LogInformation($"Processing route: {Request.Path.Value}");

            var userId =
Convert.ToInt32(HttpContext.User.FindFirstValue("Id"));
            if (userId != request.Id)
            {
                _logger.LogError($"{Request.Path.Value}: 403 Forbidden");
                return Forbid();
            }

            var user = await _mediator.Send(new
GetUserByIdRequest(request.Id));
            if (user is null)
            {
                _logger.LogError($"{Request.Path.Value}: 404 Not Found");
                return NotFound();
            }
            if (user.Password != request.OldPassword)
```

```
            {
                _logger.LogError($"{Request.Path.Value}: 400 Bad Request");
                return BadRequest();
            }
            user.Password = request.NewPassword;
            await _mediator.Send(new UpdateUserRequest(user));
            _logger.LogInformation($"{Request.Path.Value}: 200 OK");
            return Ok();
        }

        [HttpDelete("delete/id={id:int}")]
        public async Task<IActionResult> DeleteUser(int id)
        {
            _logger.LogInformation($"Processing route: {Request.Path.Value}");

            var userId =
Convert.ToInt32(HttpContext.User.FindFirstValue("Id"));
            if (userId != id)
            {
                _logger.LogError($"{Request.Path.Value}: 403 Forbidden");
                return Forbid();
            }

            await _mediator.Send(new DeleteUserRequest(id));
            _logger.LogInformation($"{Request.Path.Value}: 200 OK");
            return Ok();
        }

        [HttpGet("chats/userId={id:int}")]
        public async Task<IActionResult> GetUserChats(int id)
        {
            _logger.LogInformation($"Processing route: {Request.Path.Value}");

            var userId =
Convert.ToInt32(HttpContext.User.FindFirstValue("Id"));
            if (userId != id)
            {
                _logger.LogError($"{Request.Path.Value}: 403 Forbidden");
                return Forbid();
            }

            var chats = await _mediator.Send(new GetUserChatsRequest(id));
            var dialogues = chats?.Where(c => c.IsDialogue).ToList();

            if (dialogues is not null)
            {
                foreach (var dialogue in dialogues)
                {
                    var ids = dialogue.Name.Split('&');
                    try
                    {
```

```
                        var u1Id = Convert.ToInt32(ids[0]);
                        var u2Id = Convert.ToInt32(ids[1]);
                        int searchedId = id == u1Id ? u2Id : u1Id;
                        var user = await _mediator.Send(new
GetUserByIdRequest(searchedId));
                        dialogue.Name = user is null ? "DELETED" : user.Name;
                    }
                    catch
                    {
                        continue;
                    }
                }
            }

            var chatsDto = _mapper.Map<IEnumerable<ChatDTO>>(chats);
            _logger.LogInformation($"{Request.Path.Value}: 200 OK");
            return Ok(chatsDto);
        }

        [HttpGet("allUsers")]
        public async Task<IActionResult> GetAllUsersRequest()
        {
            _logger.LogInformation($"Processing route: {Request.Path.Value}");
            var users = await _mediator.Send(new GetAllUsersRequest());
            var usersDto = _mapper.Map<IEnumerable<UserDTO>>(users);
            _logger.LogInformation($"{Request.Path.Value}: 200 OK");
            return Ok(usersDto);
        }
    }
```

Листинг А.4 – Реализация класса ServerService

```
using Client.Domain.Entitites;
using Client.Persistence.Exceptions;
using Microsoft.AspNetCore.SignalR.Client;
using System.Net.Http.Json;
using System.Security.Cryptography;
using System.Text;

namespace Client.Persistence.Services;

internal class ServerService : IServerService
{
    private readonly HttpClient _httpClient;
    private HubConnection? _chatHubConnection;

    public event Action<Message>? GetMessageHubEvent;
    public event Action<Message>? DeleteMessageHubEvent;
    public event Action<Message>? UpdateMessageHubEvent;
    public event Action<User, Chat>? DeleteChatMemberHubEvent;
    public event Action<User, Chat>? AddChatMemberHubEvent;
    public event Action<Chat>? UpdateChatHubEvent;
```

```
        public ServerService(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        private void ConnectToHub(string token)
        {
            _chatHubConnection = new
HubConnectionBuilder().WithUrl("http://192.168.0.103:5115/messenger", options =>
            {
                options.AccessTokenProvider = () => Task.FromResult(token)!;
            })
            .Build();
            _chatHubConnection.On<Message>("SendMessage", (message) =>
            {
                GetMessageHubEvent?.Invoke(message);
            });

            _chatHubConnection.On<Message>("DeleteMessage", (message) =>
            {
                DeleteMessageHubEvent?.Invoke(message);
            });

            _chatHubConnection.On<Message>("UpdateMessage", (message) =>
            {
                UpdateMessageHubEvent?.Invoke(message);
            });

            _chatHubConnection.On<User, Chat>("DeleteChatMember", (user, chat)
=>
            {
                DeleteChatMemberHubEvent?.Invoke(user, chat);
            });

            _chatHubConnection.On<User, Chat>("AddChatMember", (user, chat) =>
            {
                AddChatMemberHubEvent?.Invoke(user, chat);
            });

            _chatHubConnection.On<Chat>("UpdateChat", (chat) =>
            {
                UpdateChatHubEvent?.Invoke(chat);
            });

            Task.Run(() => _chatHubConnection.StartAsync()).Wait();
        }

        private void DisconnectFromHub()
        {
            Task.Run(() => _chatHubConnection?.DisposeAsync()).Wait();
```

```
        }

        public List<User> GetAllUsers()
        {
            string request = $"api/User/allUsers";
            var users =
_httpClient.GetFromJsonAsync<List<User>>(request).Result;
            if (users is null)
            {
                throw new NullReferenceException("Something went wrong");
            }
            return users;
        }

        public User LoginUser(string login, string password)
        {
            User? user;
            using SHA256 hash = SHA256.Create();
            password =
Convert.ToHexString(hash.ComputeHash(Encoding.UTF8.GetBytes(password)));
            string request =
$"api/Auth/authorize/login={login}&password={password}";
            user = _httpClient.GetFromJsonAsync<User>(request).Result;
            if (user is null)
            {
                throw new NullReferenceException("No such user");
            }
            if (user.AuthorizationToken is null)
            {
                throw new Exception("Something went wrong");
            }
            _httpClient.DefaultRequestHeaders.Clear();
            _httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
user.AuthorizationToken);
            DisconnectFromHub();
            ConnectToHub(user.AuthorizationToken);
            return user;
        }

        public User RegisterUser(string username, string login, string
password)
        {
            User? user = new User() { Name = username, Login = login };
            using SHA256 hash = SHA256.Create();
            password =
Convert.ToHexString(hash.ComputeHash(Encoding.UTF8.GetBytes(password)));
            user.Password = password;
            string request = $"api/Auth/register";
            var response = _httpClient.PostAsJsonAsync(request,user).Result;
            user = response.Content.ReadFromJsonAsync<User>().Result;
            if (user is null || (int)response.StatusCode != 200)
```

```
            {
                throw new Exception("Something went wrong");
            }
            if (user.AuthorizationToken == "")
            {
                throw new Exception("Something went wrong");
            }
            _httpClient.DefaultRequestHeaders.Clear();
            _httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
user.AuthorizationToken);
            DisconnectFromHub();
            ConnectToHub(user.AuthorizationToken);
            return user;
        }

        public User UpdateUsername(User user)
        {
            string request = $"api/User/updateName";
            var response = _httpClient.PutAsJsonAsync(request, user).Result;
            var userRes = response.Content.ReadFromJsonAsync<User>().Result;
            if (userRes is null)
            {
                throw new NullReferenceException("Something went wrong");
            }
            return userRes;
        }

        public void UpdatePassword(User user, string oldPassword, string
newPassowrd)
        {
            string request = $"api/User/updatePassword";
            using SHA256 hash = SHA256.Create();
            oldPassword =
Convert.ToHexString(hash.ComputeHash(Encoding.UTF8.GetBytes(oldPassword)));
            newPassowrd =
Convert.ToHexString(hash.ComputeHash(Encoding.UTF8.GetBytes(newPassowrd)));
            var requestData = new { Id = user.Id, OldPassword = oldPassword,
NewPassword = newPassowrd};
            var response = _httpClient.PutAsJsonAsync(request,
requestData).Result;
            if ((int)response.StatusCode != 200)
            {
                throw new Exception("Something went wrong");
            }
        }

        public void DeleteUser(User user)
        {
            string request = $"api/User/delete/id={user.Id}";
            var response = _httpClient.DeleteAsync(request).Result;
        }
```

```
        public List<Chat> GetUserChats(User user)
        {
            string request = $"api/User/chats/userId={user.Id}";
            var chats =
_httpClient.GetFromJsonAsync<List<Chat>>(request).Result;
            if (chats is null)
            {
                throw new NullReferenceException("Something went wrong");
            }
            return chats;
        }

        public List<Message> GetChatMessages(Chat chat)
        {
            string request = $"api/Chat/getMessages/chatId={chat.Id}";
            List<Message>? messages = null;
            messages =
_httpClient.GetFromJsonAsync<List<Message>>(request).Result;

            if (messages is null)
            {
                throw new NullReferenceException("Something went wrong");
            }
            return messages;
        }

        public List<User> GetChatMembers(Chat chat)
        {
            string request = $"api/Chat/getMembers/chatId={chat.Id}";
            var members =
_httpClient.GetFromJsonAsync<List<User>>(request).Result;
            if (members is null)
            {
                throw new Exception("Something went wrong");
            }
            return members;
        }

        public Chat GetChatById(int id)
        {
            string request = $"api/Chat/getChat/id={id}";
            var chat = _httpClient.GetFromJsonAsync<Chat>(request).Result;
            if (chat is null)
            {
                throw new Exception("Something went wrong");
            }
            return chat;
        }

        public Message SendMessage(Message message)
```

```
        {
            string request = $"api/Chat/addMessage";
            var response = _httpClient.PostAsJsonAsync(request,
message).Result;
            var resMessage =
response.Content.ReadFromJsonAsync<Message>().Result;
            if (resMessage is null)
            {
                throw new NullReferenceException("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("SendMessage",
resMessage)).Wait();
            return resMessage;
        }

        public void DeleteMessage(Message message)
        {
            string request = $"api/Chat/deleteMessage/id={message.Id}";
            var response = _httpClient.DeleteAsync(request).Result;
            if ((int)response.StatusCode != 200)
            {
                throw new Exception("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("DeleteMessage",
message)).Wait();
        }

        public void UpdateMessage(Message message)
        {
            string request = $"api/Chat/updateMessage";
            var response = _httpClient.PutAsJsonAsync(request,
message).Result;
            var messageRes =
response.Content.ReadFromJsonAsync<Message>().Result;
            if (messageRes is null)
            {
                throw new Exception("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("UpdateMessage",
message)).Wait();
        }

        public void DeleteChatMember(Chat chat, User user)
        {
            string request =
$"api/Chat/deleteUser/userId={user.Id}&chatId={chat.Id}";
            var response = _httpClient.DeleteAsync(request).Result;
            if ((int)response.StatusCode != 200)
            {
                throw new Exception("Something went wrong");
            }
```

```
            Task.Run(() => _chatHubConnection?.InvokeAsync("DeleteChatMember",
user, chat)).Wait();
        }

        public void AddChatMember(Chat chat, User user)
        {
            string request = $"api/Chat/addUser";
            var requestData = new { UserId = user.Id, ChatId = chat.Id};
            var response = _httpClient.PostAsJsonAsync(request,
requestData).Result;
            if ((int)response.StatusCode != 200)
            {
                throw new Exception("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("AddChatMember",
user, chat)).Wait();
        }

        public void UpdateChat(Chat chat)
        {
            string request = $"api/Chat/update";
            var response = _httpClient.PutAsJsonAsync(request, chat).Result;
            var chatRes = response.Content.ReadFromJsonAsync<Chat>().Result;
            if (chatRes is null)
            {
                throw new Exception("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("UpdateChat",
chatRes)).Wait();
        }

        public Chat CreateChat(string name, User creator)
        {
            string request = $"api/Chat/create";
            var requestData = new { UserId = creator.Id, Name = name };
            var response = _httpClient.PostAsJsonAsync(request,
requestData).Result;
            var chat = response.Content.ReadFromJsonAsync<Chat>().Result;
            if (chat is null)
            {
                throw new Exception("Something went wrong");
            }
            return chat;
        }

        public Chat CreateDialogue(User user1, User user2)
        {
            string request = $"api/Chat/createDialogue";
            var requestData = new
            {
                User1Id = user1.Id,
```

```
                User1Name = user1.Name,
                User2Id = user2.Id,
                User2Name = user2.Name
            };
            var response = _httpClient.PostAsJsonAsync(request,
requestData).Result;
            if ((int)response.StatusCode == 400)
            {
                var dialogueId = response.Content.ReadAsStringAsync().Result;
                if (dialogueId is not null)
                {
                    throw new DialogueExistException(dialogueId);
                }
                else
                {
                    throw new Exception("Something went wrong");
                }
            }
            var chat = response.Content.ReadFromJsonAsync<Chat>().Result;
            if (chat is null)
            {
                throw new Exception("Something went wrong");
            }
            Task.Run(() => _chatHubConnection?.InvokeAsync("AddChatMember",
user1, chat)).Wait();
            Task.Run(() => _chatHubConnection?.InvokeAsync("AddChatMember",
user2, chat)).Wait();
            return chat;
        }
    }
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Схемы алгоритмов программного средства

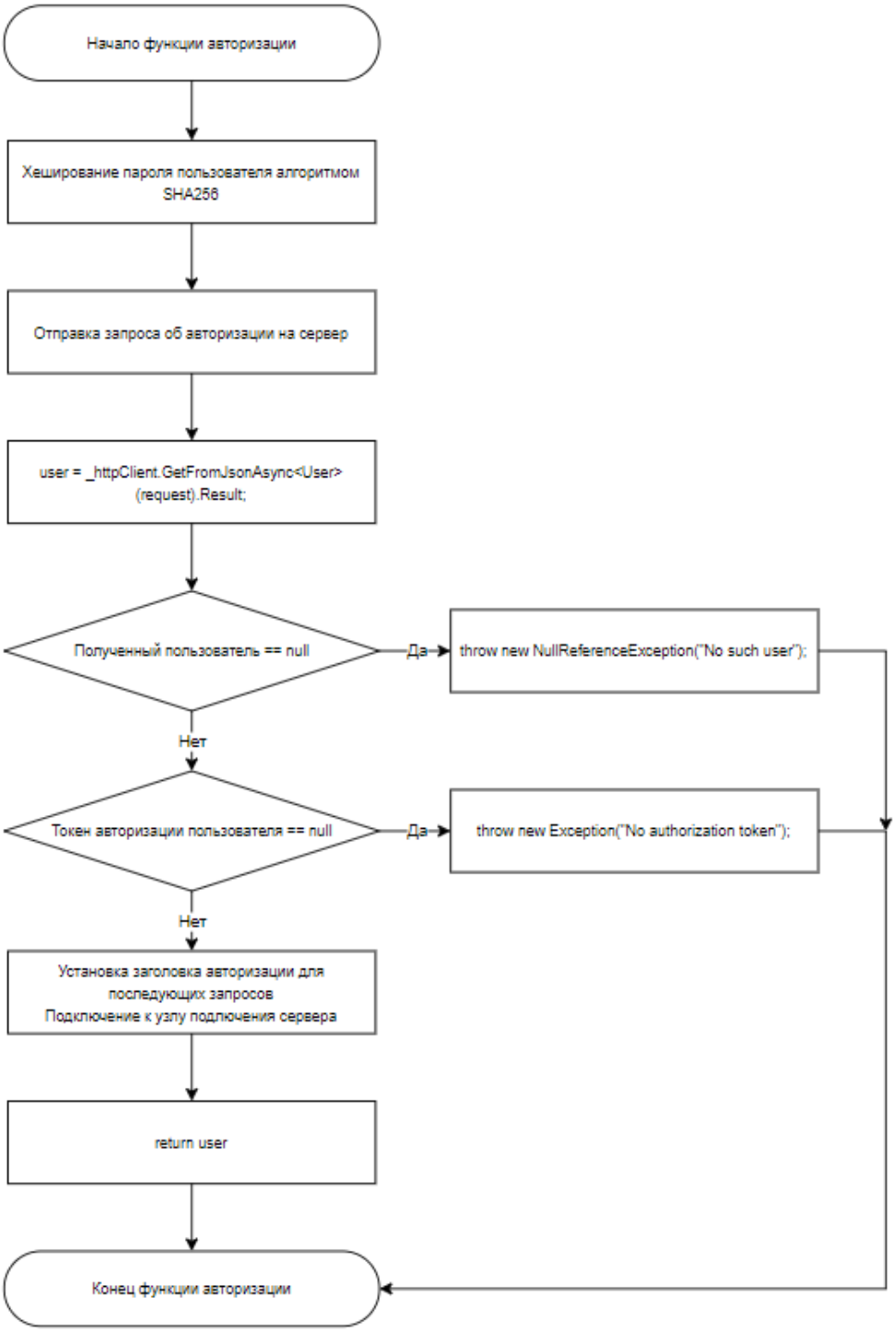

Рисунок Б.1 – Блок-схема функции авторизации в клиентском приложении

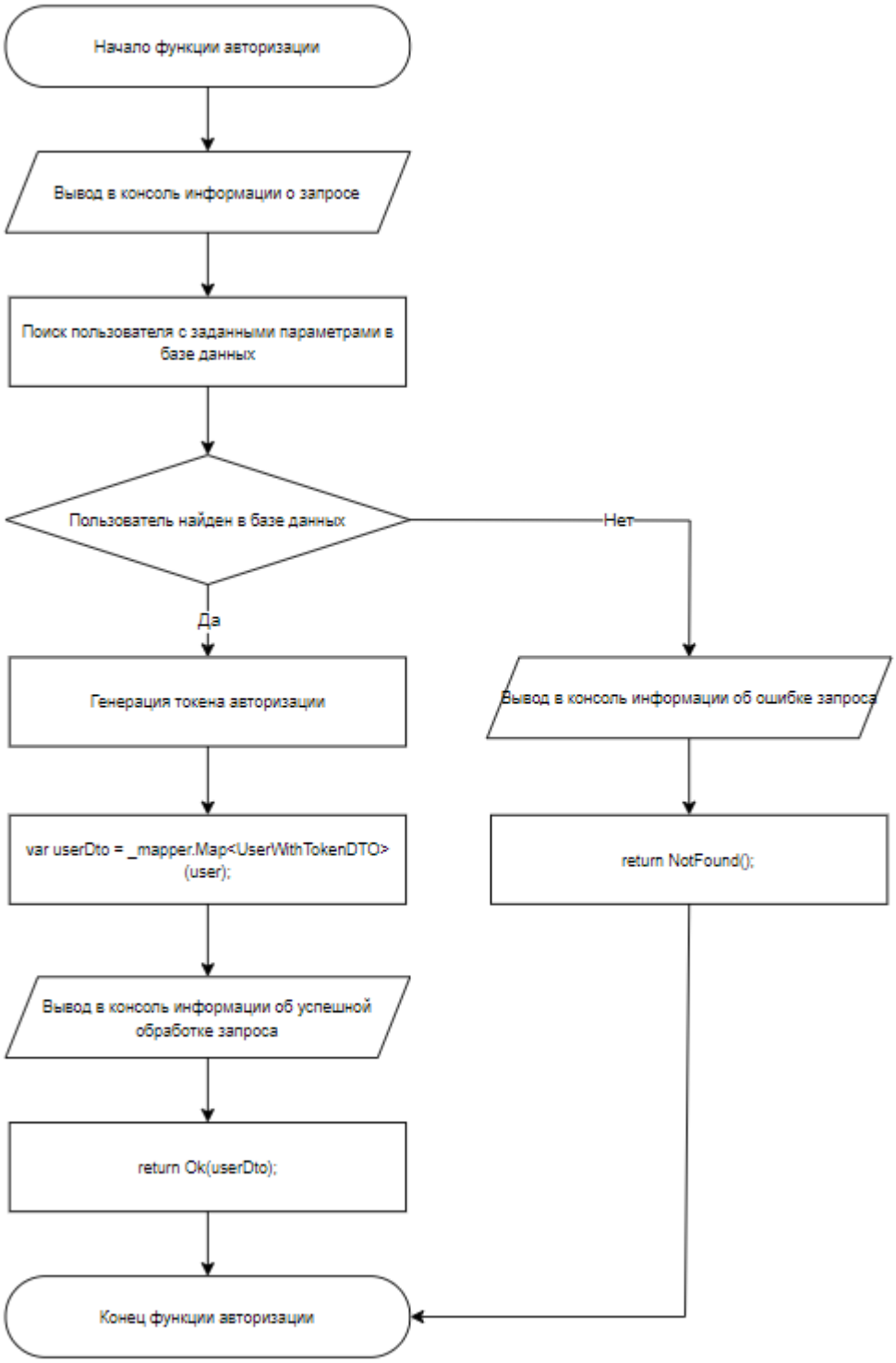

Рисунок Б.2 – Блок-схема функции авторизации в серверном приложении

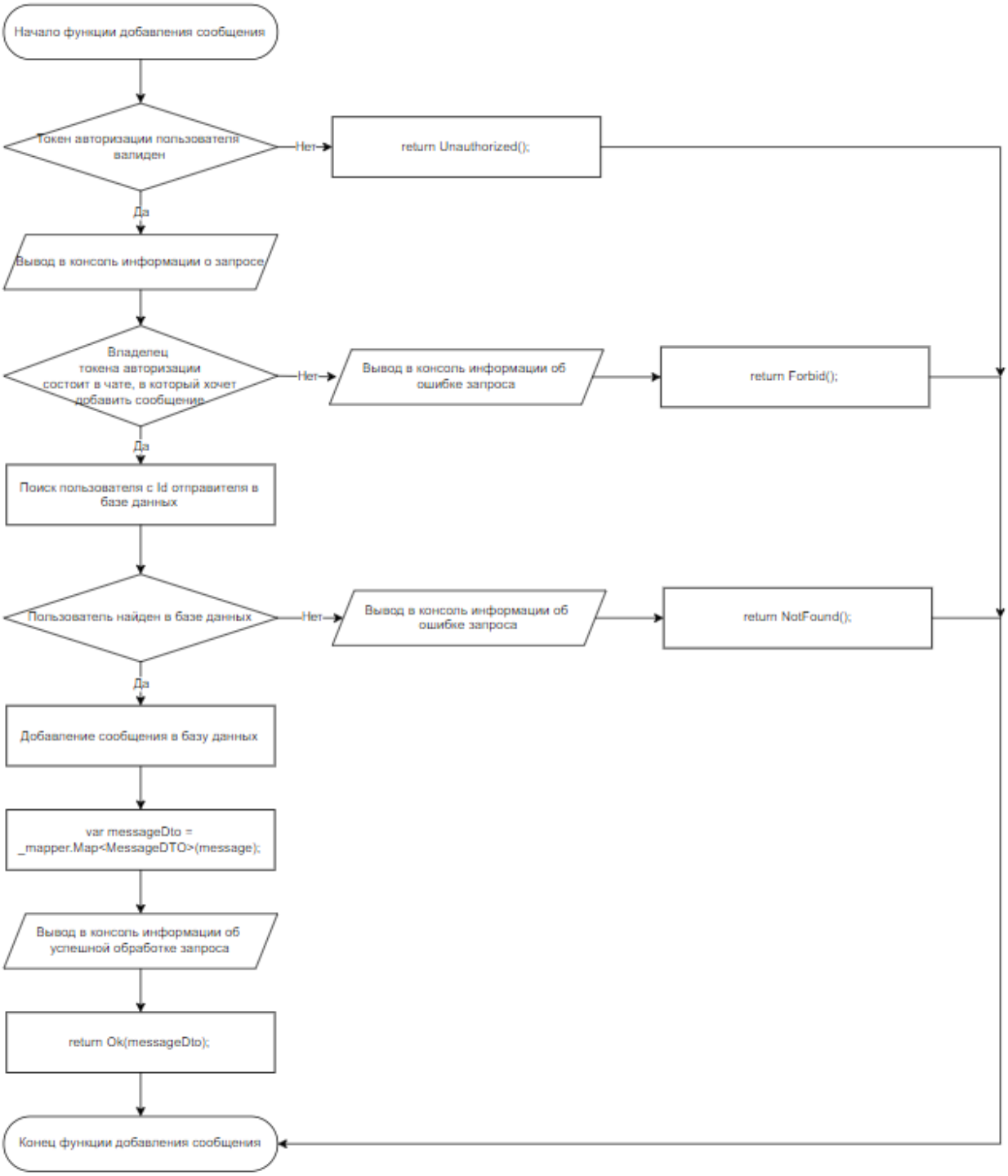

Рисунок Б.3 – Блок-схема функции добавления сообщения в серверном приложении

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1-40 04 01	Пояснительная записка	45 с.
	Графические документы	
ГУИР 253505 015 СА	Программное средство для обмена сообщениями с функцией создания многопользовательских чатов. Схемы алгоритмов	Формат А4

					БГУИР КП 1-40 04 01			
Изм.	Л.	№ докум.	Подп.	Дата	Программное средство для обмена сообщениями с функцией создания многопользовательских чатов		Лист	Листов
Разраб.		*Павлович В. Ю.*		04.06.24		Т	45	45
Пров.		*Тушинская Е.В.*		04.06.24	Ведомость курсового проекта	Кафедра Информатики гр. 253505		