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

**CОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc168004270)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc168004271)

[1.1 Обзор аналогов 4](#_Toc168004272)

[1.2 Постановка задачи 4](#_Toc168004273)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 5](#_Toc168004274)

[2.1 Общая информация 5](#_Toc168004275)

[2.2 Разработка функциональности программного средства 6](#_Toc168004276)

[2.3 Архитектура программного средства 7](#_Toc168004277)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 9](#_Toc168004278)

[3.1 Разработка серверного приложения 9](#_Toc168004279)

[3.1.1 Разработка слоя представления 9](#_Toc168004280)

[3.1.2 Разработка слоя приложения 10](#_Toc168004281)

[3.1.3 Разработка слоя инфраструктуры 11](#_Toc168004282)

[3.1.4 Разработка доменного слоя 12](#_Toc168004283)

[3.2 Разработка клиентского приложения 12](#_Toc168004284)

[3.2.1 Разработка слоя представления 12](#_Toc168004285)

# ВВЕДЕНИЕ

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

### 1.1 Обзор аналогов

**//ТЕКСТ ПРО ОБЗОР АНАЛОГОВ**

### 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 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. С# 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 С#, 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 USE-CASE ДИАГРАММЫ**

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

## 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 инфраструктуры

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

## 3.2.1 Разработка слоя представления