

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Программирование сетевых приложений»
на тему: «Учебная криптобиржа»

Исполнитель: студент гр. ИТП-41
Дроздов Д.С.

Руководитель: преподаватель
Гуменников Е. Д.

Дата проверки: _____

Дата допуска к защите: _____

Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2024

СОДЕРЖАНИЕ

Введение	5
1 Обзор программных средств для решения поставленной задачи	6
1.1 Сетевые протоколы	6
1.2 Обзор протокола <i>TCP</i> для передачи данных по сети	7
1.3 Клиент-серверная архитектура	8
1.3 Язык программирования <i>C#</i> и база данных <i>PostgreSQL</i>	10
2 Архитектура и структура приложения.....	13
2.1 Компоненты приложения.....	13
2.2 Классы приложения	14
3 Результаты верификации программного обеспечения.....	20
3.1 Описание пользовательского интерфейса.....	20
3.2 Верификация программного обеспечения.....	24
Заключение	27
Список использованных источников	28
Приложение А	29
Приложение Б	30
Приложение В.....	31
Приложение Г	32
Приложение Д.....	90

ВВЕДЕНИЕ

В современном мире криптовалюты становятся всё более значимой частью финансового сектора, привлекая внимание как профессиональных трейдеров, так и начинающих инвесторов. Однако торговля реальными криптоактивами сопряжена с существенными рисками потери средств, особенно для неопытных пользователей. В связи с этим возникает потребность в создании учебных платформ, позволяющих безопасно практиковать навыки трейдинга без использования реальных денежных средств.

Актуальность разработки учебной криптобиржи обусловлена растущим интересом к криптовалютному рынку и необходимостью предоставления безопасной среды для обучения трейдингу. Такая платформа позволит пользователям освоить принципы работы криптовалютного рынка, изучить различные торговые стратегии и получить практический опыт без финансовых рисков.

Целью данной курсовой работы является разработка веб-приложения «Учебная криптобиржа», предоставляющего пользователям возможность практиковать навыки криптовалютной торговли в безопасной среде с использованием виртуальных активов.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать систему регистрации и аутентификации пользователей;
- реализовать функционал управления виртуальным балансом и торговыми операциями;
- интегрировать получение реальных рыночных котировок через *WebSocket* протокол;
- создать систему маржинальной торговли с настраиваемым плечом;
- разработать инструменты технического анализа и визуализации данных;
- обеспечить защиту передаваемых данных с использованием современных протоколов шифрования.

В процессе разработки будут использованы современные технологии и методологии программирования, обеспечивающие высокую производительность и надежность приложения. Особое внимание будет уделено безопасности данных и удобству пользовательского интерфейса.

1 ОБЗОР ПРОГРАММНЫХ СРЕДСТВ ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Сетевые протоколы

Сетевые протоколы – это набор правил, которые определяют, как устройства и программы обмениваются данными в компьютерных сетях. Они обеспечивают совместимость и эффективность взаимодействия между различными системами. Протоколы выполняют разные функции: от установления соединения до передачи данных и управления ошибками.

Протоколы канального уровня отвечают за физическую передачу данных по сети и определяют, как устройства в одной локальной сети обмениваются информацией. Например, *Ethernet* – это один из самых распространенных протоколов, используемый в локальных сетях. Он определяет, как данные должны быть упакованы и переданы по кабелям сети. *Wi-Fi*, в свою очередь, используется для передачи данных по радиосигналам в беспроводных локальных сетях [1, с. 78].

Сетевые протоколы управляют маршрутизацией данных между разными сетями. Основным протоколом этой категории является *IP (Internet Protocol)*. Он обеспечивает адресацию и маршрутизацию данных, позволяя устройствам находить друг друга и передавать пакеты по сети. *IP* существует в двух версиях: *IPv4*, который используется уже много лет, и *IPv6*, созданный для решения проблемы нехватки уникальных *IP*-адресов.

Протоколы транспортного уровня отвечают за надежную передачу данных между узлами сети. Примером такого протокола является *TCP (Transmission Control Protocol)*. *TCP* обеспечивает надежную передачу данных, подтверждая получение пакетов и повторно отправляя их в случае потерь. Это особенно важно для приложений, где важна целостность данных, таких как веб-браузеры и электронная почта.

Протокол *WebSocket* относится к протоколам транспортного уровня. Он работает поверх протокола *TCP*, обеспечивая двустороннюю, полноценную коммуникацию в реальном времени между клиентом и сервером. *WebSocket* позволяет установить постоянное соединение, в отличие от традиционного *HTTP*, где каждое взаимодействие требует нового запроса и ответа. Это соединение сохраняется открытым, позволяя обмениваться данными без необходимости повторного установления соединения, что делает его идеальным для приложений, где требуется постоянная передача данных, таких как чаты, онлайн-игры или финансовые приложения.

UDP (User Datagram Protocol) – протокол без установления соединения, который работает быстрее, но не гарантирует доставку данных. *UDP*

используется в ситуациях, где скорость важнее надежности, например, в онлайн-играх и видеоконференциях.

Протоколы прикладного уровня обеспечивают взаимодействие между приложениями и пользователями. Например, *HTTP* (*Hypertext Transfer Protocol*) используется для передачи данных в интернете, когда вы заходите на веб-сайт. Его защищенная версия, *HTTPS*, шифрует данные для их безопасной передачи.

Протоколы уровня сеанса управляют установлением, поддержанием и завершением сеансов связи между программами. Например, *RPC* (*Remote Procedure Call*) позволяет программам вызывать функции на удаленных системах. *NetBIOS* (*Network Basic Input/Output System*) используется для взаимодействия в локальных сетях, особенно в старых системах *Windows*.

Протоколы уровня представления отвечают за форматирование и представление данных. Протоколы такие как *TLS/SSL* (*Transport Layer Security / Secure Sockets Layer*) обеспечивают шифрование данных, защищая их от несанкционированного доступа.

1.2 Обзор протокола *TCP* для передачи данных по сети

Протокол *TCP* представляет собой один из основных компонентов стека протоколов интернета. Это надежный протокол транспортного уровня, который обеспечивает установление, поддержание и завершение соединений между узлами сети. *TCP* предназначен для гарантированной доставки данных в правильной последовательности, что делает его ключевым элементом для приложений, требующих высокой точности передачи, таких как онлайн-обучение, видео-конференции и обмен сообщениями [1, с. 314].

Работа *TCP* начинается с процесса установки соединения между двумя узлами, обычно называемого трехсторонним рукопожатием. Этот процесс состоит из обмена специальными служебными сообщениями, которые обеспечивают согласование параметров соединения, таких как размер окна передачи данных, и гарантируют, что обе стороны готовы к взаимодействию. После установления соединения начинается передача данных. Пример схемы работы рукопожатия представлен на рисунке 1.1.

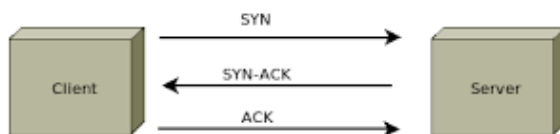


Рисунок 1.1 – Схема рукопожатия

Данные, которые передаются через *TCP*, разбиваются на небольшие сегменты, которые отправляются на другой конец соединения. Каждый сегмент сопровождается заголовком, содержащим служебную информацию, такую как номер последовательности и контрольная сумма. Номер последовательности позволяет получателю определить порядок, в котором должны быть собраны сегменты, а контрольная сумма обеспечивает проверку целостности данных. Если какой-либо сегмент теряется или повреждается, *TCP* автоматически инициирует его повторную передачу. Этот процесс делает передачу данных надежной даже в условиях нестабильного соединения.

По мере получения сегментов принимающая сторона отправляет подтверждения отправителю, информируя его о том, что данные получены корректно. Этот механизм называется управлением потоком. Он также включает в себя регулирование скорости передачи данных, чтобы предотвратить перегрузку сети или конечного устройства. *TCP* использует динамическую настройку размера окна передачи данных, которая позволяет адаптироваться к текущей пропускной способности сети и обеспечивает эффективное использование ресурсов.

Когда передача данных завершена, *TCP* завершает соединение с помощью последовательности обмена сообщениями, которая гарантирует, что обе стороны полностью передали и получили всю необходимую информацию. Этот процесс называется завершением соединения, и он также выполняется надежно и синхронно, чтобы избежать потерь данных.

1.3 Клиент-серверная архитектура

Клиент-серверная архитектура – это модель взаимодействия между двумя основными компонентами: клиентом и сервером. Она широко используется в разработке распределённых систем и веб-приложений, обеспечивая разделение задач и упрощая управление данными и логикой [2].

Клиент – это компонент, который отправляет запросы серверу и ожидает ответы. Он может быть программным обеспечением, например, веб-браузером, мобильным приложением или десктопным приложением. Клиентская сторона может быть как простым интерфейсом для пользователя, так и более сложным приложением, выполняющим определённую логику.

Сервер – это компонент, который принимает запросы от клиентов, обрабатывает их и возвращает ответы. Серверы могут быть специализированными для выполнения определённых функций, таких как базы данных, веб-серверы, почтовые серверы и т.д. Они управляют хранением данных, выполняют вычислительные задачи или обрабатывают запросы по бизнес-логике. Серверы часто имеют более мощные вычислительные ресурсы, чем клиенты.

Принципы работы клиент-серверной архитектуры включают взаимодействие по модели запроса-ответа, где клиент отправляет запрос серверу, а сервер обрабатывает его и возвращает ответ. Этот принцип лежит в основе взаимодействия веб-браузеров с веб-серверами, где клиент запрашивает веб-страницу, а сервер возвращает её в виде *HTML*-разметки. В клиент-серверной модели также важным является разделение логики: клиент занимается пользовательским интерфейсом, а сервер управляет данными и выполняет основную бизнес-логику. Это упрощает разработку и поддержание системы, так как каждая сторона может развиваться независимо.

Сетевое взаимодействие является основой клиент-серверной архитектуры. Клиент и сервер взаимодействуют через сеть (интернет, локальную сеть и т.д.), используя различные протоколы, такие как *HTTP/HTTPS*, *TCP/IP*, *WebSocket* и другие. Это обеспечивает возможность обмена данными и командной синхронизации между компонентами, находящимися в разных географических точках.

Одним из ключевых аспектов клиент-серверной архитектуры является идентификация и аутентификация, когда сервер требует от клиента подтверждения его прав доступа. Это обеспечивает безопасность данных и защищает ресурсы от несанкционированного доступа. Для этого используются логин и пароль, токены, сертификаты и двухфакторная аутентификация.

Клиент-серверная архитектура имеет множество преимуществ. Одним из них является централизованное управление, при котором сервер управляет данными и логикой приложения, упрощая администрирование и безопасность. Также архитектура поддерживает масштабируемость, что позволяет добавлять новые серверы для обработки увеличенного трафика и нагрузки. Разделение логики между клиентом и сервером обеспечивает переиспользуемость кода, так как серверная логика может использоваться разными клиентами. Это упрощает разработку новых приложений, использующих те же серверные ресурсы.

Клиент-серверная архитектура используется в различных приложениях. Веб-приложения, мобильные приложения, многопользовательские игры и финансовые системы – все они используют этот принцип для обмена данными и выполнения операции.

В курсовом проекте, основанном на клиент-серверной архитектуре, можно разработать клиентскую часть в виде веб-приложения, которая обеспечит пользователю удобный интерфейс и взаимодействие. Серверная же часть, будет обрабатывать запросы от клиента, выполнять запросы к базе данных, возвращать пользователю определенные данные и так далее.

1.3 Язык программирования C# и база данных *PostgreSQL*

1.3.1 C# – это объектно-ориентированный язык программирования, разработанный корпорацией *Microsoft* в рамках платформы *.NET*. Он широко используется для создания различных приложений: от десктопных до веб-приложений, мобильных платформ, игр и сложных серверных решений [3, с. 23].

Одной из ключевых особенностей языка является его тесная интеграция с платформой *.NET*, которая предоставляет разработчикам доступ к обширной библиотеке классов и инструментов для работы с файлами, сетью, базами данных и другими ресурсами.

C# поддерживает принципы объектно-ориентированного программирования, такие как наследование, полиморфизм, инкапсуляция и абстракция, что делает его удобным для разработки сложных и масштабируемых систем. Асинхронное программирование является ещё одной важной особенностью языка. Используя ключевые слова *async* и *await*, разработчики могут выполнять длительные операции, такие как запросы к базам данных или внешним *API*, без блокировки основного потока выполнения, что особенно важно для серверных приложений. С развитием языка в последних версиях C# появились такие современные функции, как стрелочные функции, шаблоны сопоставления, кортежи, записи (*records*) и модули, что значительно упростило разработку и сделало язык более гибким и выразительным.

1.3.2 ASP.NET – это мощный и гибкий фреймворк для разработки веб-приложений, созданный корпорацией *Microsoft*. Он является частью платформы *.NET* и предоставляет все необходимые инструменты для создания современных, высокопроизводительных и масштабируемых веб-приложений. *ASP.NET* поддерживает разработку серверных приложений (включая *RESTful API*), веб-интерфейсов, в реальном времени и даже облачных решений, что делает его универсальным выбором для различных типов проектов. [4, с. 22].

Фреймворк также предоставляет развитую систему маршрутизации. Разработчики могут настроить маршруты для обработки запросов, определяя, какие *URL* должны вызывать определенные методы контроллеров или страницы *Razor*. Поддержка маршрутов с параметрами и атрибутная маршрутизация позволяет легко управлять логикой обработки запросов.

С момента своего появления *ASP.NET* претерпел значительные изменения. Современная версия *ASP.NET Core* – представляет собой кроссплатформенный, модульный и открытый фреймворк, который работает на *Windows*, *Linux* и *macOS*. Это позволяет создавать приложения, которые могут быть развернуты на любых операционных системах, включая облачные платформы, такие как *Azure*, *AWS* или *Google Cloud*. Пример архитектуры приложения *ASP.NET Core* представлен на рисунке 1.2.

ASP.NET Core Architecture

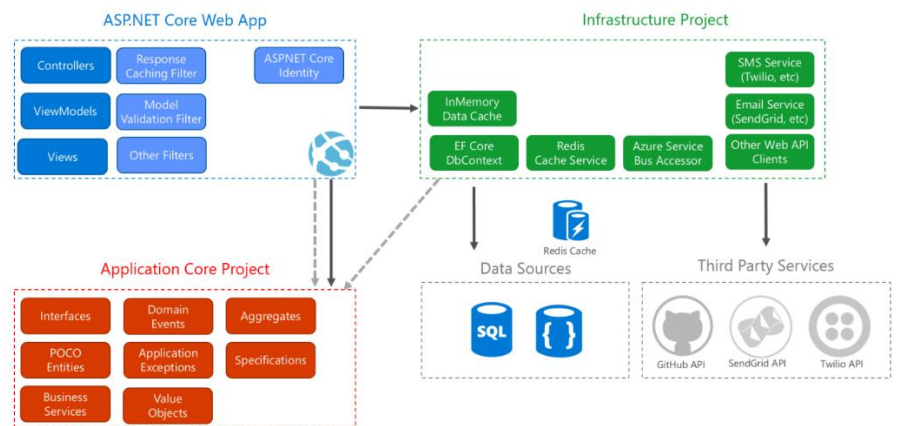


Рисунок 1.2 – Архитектуры *ASP.NET Core*

В рамках разработки приложений в реальном времени *ASP.NET Core* предоставляет поддержку *SignalR* – технологии для двусторонней связи между клиентом и сервером. *SignalR* позволяет реализовать такие функции, как чаты, системы уведомлений и обновление данных в реальном времени, без необходимости вручную управлять *WebSocket*-соединениями.

Одной из ключевых особенностей *ASP.NET* является его модульная архитектура. Разработчики могут использовать только те компоненты, которые необходимы для их проекта, что позволяет минимизировать размер приложения и повысить его производительность. Например, можно подключить дополнительные модули для работы с аутентификацией, сжатием данных, маршрутизацией или обработкой статических файлов.

1.3.3 PostgreSQL – это одна из самых мощных и надежных объектно-реляционных систем управления базами данных. *PostgreSQL* является СУБД с открытым исходным кодом, что делает её доступной для бесплатного использования. Она известна своей производительностью, соответствием стандарту *SQL* и поддержкой расширенной функциональности. *PostgreSQL* поддерживает транзакции с использованием механизма *ACID*, что гарантирует надежность и целостность данных даже в сложных системах. База данных также предоставляет широкий набор индексов, включая *Hash*, *GIN* и *GiST*, что позволяет эффективно обрабатывать сложные запросы и работать с большими объемами данных. [5].

Одной из ключевых особенностей *PostgreSQL* является поддержка *JSON* и *JSONB*, что позволяет использовать её как гибридную реляционную и документную базу данных. Это удобно для приложений, которые работают с полуструктурированными данными. *PostgreSQL* также поддерживает пользовательские функции, триггеры, хранимые процедуры и расширения, такие как *PostGIS* для работы с географическими данными или *TimescaleDB* для временных рядов. Это

делает её гибким инструментом для работы с различными типами данных и аналитическими задачами.

PostgreSQL отличается своей кроссплатформенностью и масштабируемостью. Она может работать на *Windows*, *Linux* и *macOS*, а также поддерживает как вертикальную (увеличение мощности сервера), так и горизонтальную масштабируемость (репликация и шардирование). Благодаря этому *PostgreSQL* является отличным выбором для высоконагруженных систем и сложных приложений. *PostgreSQL* предоставляет все необходимые инструменты для хранения данных о пользователях, их балансах, истории сделок и работе с реальными рыночными котировками. Она также позволяет выполнять сложные аналитические запросы для обработки данных и прогнозирования рыночных трендов.

Таким образом, связка языка программирования *C#* и базы данных *PostgreSQL* является отличным выбором для разработки серверной части учебной криптобиржи. *C#* обеспечивает высокую производительность, удобство разработки и возможность интеграции с другими технологиями, а *PostgreSQL* предоставляет надежное и эффективное управление данными. Вместе они создают мощную основу для создания сложных и масштабируемых приложений, обеспечивая высокую производительность и безопасность.

2 АРХИТЕКТУРА И СТРУКТУРА ПРИЛОЖЕНИЯ

2.1. Компоненты приложения

Клиент-серверное приложение включает две основные составляющие: клиент и сервер. Сервер представляет собой приложение, которое обрабатывает запросы клиента. Взаимодействие между клиентом и сервером осуществляется через протокол *HTTP*, реализованный поверх протокола *TCP*. Это требует, чтобы серверное приложение умело корректно обрабатывать входящие *HTTP*-запросы, а также формировать соответствующие заголовки ответов.

Для реализации эффективного взаимодействия между клиентом и сервером необходимо создать механизм роутинга на сервере. Этот механизм будет сопоставлять определённые действия с ресурсами, запрошенными пользователем. Например, при получении *HTTP*-запроса сервер должен корректно распознать запрос, обработать его и выдать клиенту необходимый ресурс.

Ключевым моментом является выбор грамотной архитектуры серверного приложения. В контексте клиент-серверного взаимодействия отлично подходит архитектура *REST*. *REST* представляет собой архитектурный стиль, основанный на принципах проектирования, которые обеспечивают упрощение взаимодействия и масштабируемость системы.

Каждая сущность приложения в архитектуре *REST* представлена в виде ресурса. В рамках реализации сервиса онлайн-обучения необходимо создать ресурсы для работы с сущностями: *User*, *Trade*, *Notification*, *Token*, *Order*.

Также обязательным элементом приложения является система аутентификации пользователей. Для этого можно использовать *JWT (JSON Web Token)*, который позволяет безопасно идентифицировать пользователя и передавать его данные между клиентом и сервером.

Клиентское приложение реализовано с помощью фреймворка *Blazor WebAssembly (WASM)* должно предоставлять пользователю возможность настройки начального количества валюты, после чего пользователи смогут открывать сделки с маржинальным плечом на различные криптовалютные пары, используя реальные котировки, а также поддерживать основные функции управления аккаунтом, такие как пополнение виртуального баланса, просмотр истории сделок и анализ текущих позиций. Пользователи должны иметь возможность создавать учетные записи, настраивать параметры торговли и получать уведомления о важных событиях на рынке. Для обработки всех этих операций на сервере должны быть реализованы соответствующие сущности, сервисы и контроллеры.

2.2 Классы приложения

В проекте реализована клиент-серверная архитектура с разделением на:
Backend:

- *REST API* на *ASP.NET Core*;
- работа с базой данных через *Entity Framework Core*;
- система аутентификации и авторизации через *JWT* токены;
- бизнес-логика в сервисах;
- взаимодействие с внешними *API* для получения рыночных данных.

Frontend (Blazor):

- пользовательский интерфейс на *Blazor*;
- *MVVM* архитектура для *UI* компонентов;
- Компонентный подход к построению интерфейса;
- Взаимодействие с *backend* через *HTTP*.

Модели данных:

DepositRequestDto – модель для запросов на внесение депозита, которая содержит следующие поля:

- *amount*: десятичное значение, представляющее сумму депозита, должна быть больше 0.01;
- *asset*: строка, указывающая тип актива для депозита.

HistoricalData – сущность для хранения исторических данных по ценам активов, которая содержит следующие поля:

- *Id*: целочисленный уникальный идентификатор для записи исторических данных;
- *pair*: строка, представляющая торговую пару;
- *timestamp*: объект *DateTime*, отмечающий время записи цены;
- *price*: десятичное значение, указывающее цену актива на заданный момент времени.

LoginDto – модель данных для входа пользователя в систему, которая содержит следующие поля:

- *email*: строка для электронной почты пользователя;
- *password*: строка для пароля пользователя.

MarketData – сущность для хранения текущей рыночной информации о ценах криптовалют, которая содержит следующие поля:

- *Id*: целое число для уникального идентификатора записи рыночных данных;
- *pair*: строка, указывающая торговую пару;
- *price*: десятичное значение текущей цены криптовалюты;
- *timestamp*: объект *DateTime*, по умолчанию установлен на текущее время по *UTC*, указывающий на момент фиксации цены.

OpenTradeRequestDto – модель для запроса на открытие новой торговой сделки, которая содержит следующие поля:

- pair*: строка для торговой пары;
- type*: строка, обозначающая тип сделки ('buy' или 'sell');
- leverage*: целое число для указания кредитного плеча;
- amount*: десятичное значение объема сделки;
- stopLoss*: опциональное десятичное значение для уровня стоп-лосса;
- takeProfit*: опциональное десятичное значение для уровня тейк-профита.

Notification – сущность для уведомлений пользователя, которая содержит следующие поля:

- Id*: целочисленный идентификатор уведомления;
- userId*: строка, являющаяся внешним ключом, связанным с пользователем;
- message*: строка с текстом уведомления;
- isRead*: логическое значение, показывающее, было ли уведомление прочитано (по умолчанию *false*);
- createdAt*: объект *DateTime*, показывающий время создания уведомления, по умолчанию *UTC now*;
- user*: ссылка на объект пользователя.

Order – модель для представления заказов на покупку или продажу, которая содержит следующие поля:

- Id*: целое число, уникальный идентификатор заказа;
- userId*: строка внешний ключ на пользователя;
- pair*: строка, представляющая торговую пару;
- type*: строка, указывающая тип операции ('buy' или 'sell');
- amount*: Десятичное значение объема заказа;
- price*: Десятичное значение цены заказа;
- status*: Строка статуса заказа (по умолчанию *'pending'*);
- createdAt*: объект *DateTime*, показывающий время создания заказа, по умолчанию *UTC now*;
- user*: ссылка на объект пользователя.

OrderBookResponse – модель для ответа с книгой ордеров, которая содержит следующие поля:

- lastUpdateId*: длинное целое число, указывающее последний *ID* обновления;
- bids*: список списков строк, представляющих предложения купить;
- asks*: список списков строк, представляющих предложения продать.

RegisterDto – модель для регистрации нового пользователя, которая содержит следующие поля:

- email*: строка для *email* нового пользователя;

–*password*: строка для пароля нового пользователя.

Token – модель для описания криптовалютных токенов, которая содержит следующие поля:

- Id*: целое число уникальный идентификатор токена;
- name*: строка с названием токена (например, *BTC*, *ETH*);
- symbol*: строка с символом токена (например, *BTC*, *ETH*);
- userAssets*: коллекция объектов *UserAsset*, связанных с токеном.

Trade – сущность для представления торговых сделок, которая содержит следующие поля:

- Id*: целое число уникальный идентификатор сделки;
- userId*: строка внешний ключ на пользователя;
- pair*: строка, представляющая торговую пару;
- type*: строка, указывающая тип сделки ('buy' или 'sell');
- leverage*: целое число для указания кредитного плеча;
- amount*: десятичное значение объема сделки;
- entryPrice*: десятичное значение цены открытия сделки;
- exitPrice*: опциональное десятичное значение цены закрытия сделки;
- stopLoss*: опциональное десятичное значение для уровня стоп-лосса;
- takeProfit*: опциональное десятичное значение для уровня тейк-профита;
- profitLoss*: опциональное десятичное значение прибыли/убытка;
- status*: строка статуса сделки ('open' или 'closed');
- openedAt*: объект *DateTime* времени открытия сделки;
- closedAt*: опциональный объект *DateTime* времени закрытия сделки;
- fee*: опциональное десятичное значение комиссии;
- user*: ссылка на объект пользователя.

User – модель пользователя в системе, которая содержит следующие поля:

- balance*: десятичное значение начального баланса пользователя;
- createdAt*: объект *DateTime* времени регистрации;
- lastLoginAt*: опциональный объект *DateTime* последнего входа;
- trades*: коллекция связанных сделок;
- userAssets*: коллекция активов пользователя;
- notifications*: коллекция уведомлений для пользователя.

UserAsset – сущность для связи пользователя с его активами, которая содержит следующие поля:

- Id*: целое число уникальный идентификатор;
- userId*: строка внешний ключ на пользователя;
- asset*: строка с названием актива (например, *BTC*, *USDT*);
- balance*: десятичное значение баланса актива;
- updatedAt*: объект *DateTime* времени последнего обновления баланса;
- user*: ссылка на объект пользователя;

–*token*: ссылка на объект токена;

Контроллеры:

AuthController – управление аутентификацией:

- регистрация новых пользователей;
- аутентификация и выдача *JWT* токенов;
- управление пользовательскими сессиями.

AssetController – управление активами:

- получение списка активов пользователя;
- операции пополнения баланса;
- работа с токенами.

TradeController – управление торговлей:

- создание и управление сделками;
- взаимодействие с рыночными данными;
- обработка торговых операций.

NotificationController – система уведомлений:

- управление уведомлениями пользователя;
- отметка о прочтении;
- удаление уведомлений.

ChartController – работа с графиками:

- получение исторических данных;
- обработка рыночной информации.

Сервисы:

JwtTokenService – сервис для работы с *JWT* токенами:

- генерация токенов для аутентификации;
- настройка параметров токена (время жизни, издатель, аудитория);
- добавление *claims* в токен.

MarketService (IMarketService) – сервис для работы с рыночными данными:

- получение текущих цен криптовалют;
- получение исторических данных;
- работа с *order book*;
- кэширование данных.

NotificationService (INotificationService) – сервис уведомлений:

- создание уведомлений для пользователей;
- массовая рассылка уведомлений;
- сохранение уведомлений в базе данных.

HttpRequestService (IHttpRequestService) – сервис для *HTTP* запросов:

- обобщенный метод для *GET* запросов;
- обработка ошибок *HTTP*;
- десериализация ответов.

CacheService (ICacheService) – сервис кэширования:

- получение данных из кэша;
- сохранение данных в кэш;
- управление временем жизни кэша.

AuthService – это сервис, отвечающий за взаимодействие с сервером для выполнения всех операций, связанных с аутентификацией и авторизацией пользователя. Он управляет входом и регистрацией пользователей, проверяет их текущее состояние (аутентифицированы они или нет), а также работает с токенами. При входе в систему сервис отправляет запрос на сервер с данными пользователя (например, *email* и пароль). Если сервер подтверждает их корректность, сервис получает токен, сохраняет его в локальном хранилище и уведомляет приложение о входе пользователя. При регистрации новый пользователь добавляется в систему, а сервер возвращает сообщение об успешной регистрации или ошибке. Для проверки состояния аутентификации *AuthService* извлекает токен из локального хранилища и анализирует его. Если токен истек или отсутствует, пользователь считается неаутентифицированным. Также сервис поддерживает обновление токена с использованием *refresh*-токена, если основной токен стал недействительным. При выходе из системы токены удаляются, а приложение уведомляется о выходе. Кроме того, сервис может извлечь из токена имя пользователя, например, чтобы отобразить его в интерфейсе.

Класс *CustomAuthenticationStateProvider* управляет состоянием аутентификации в приложении и отвечает за то, чтобы приложение знало, вошел пользователь в систему или нет. Когда пользователь входит, провайдер парсит токен, чтобы извлечь из него информацию, такую как роли и *email* пользователя, и обновляет состояние аутентификации. Это позволяет интерфейсу приложения адаптироваться под текущего пользователя (например, показывать его имя или предоставлять доступ к определенным функциям для авторизованных пользователей). Если пользователь выходит из системы, провайдер очищает состояние аутентификации, делая пользователя анонимным. В случае, если токен недействителен или отсутствует, провайдер также возвращает состояние анонимного пользователя. Все изменения состояния передаются приложению, чтобы оно могло обновить пользовательский интерфейс.

Интерфейс *IAAuthService* определяет контракт, который должен реализовать любой сервис, занимающийся аутентификацией. Он описывает методы для входа, регистрации, проверки состояния аутентификации, выхода из системы, получения токена и отображаемого имени пользователя. Это позволяет легко заменить реализацию *AuthService*, если потребуется, не изменяя остальной код приложения.

Интерфейс *ITradeService* описывает функционал для работы с торговыми операциями. Он используется для получения списка активных сделок, создания

новых сделок, закрытия существующих и просмотра истории завершенных операций. Реализация этого интерфейса взаимодействует с сервером, чтобы отправлять и получать данные о сделках.

DTO-классы, такие как *LoginRequest*, *RegisterRequest*, *AuthResponse*, *UserDto*, *TradeDto* и *TradeRequest*, используются для передачи данных между клиентом и сервером. Например, при входе данные пользователя передаются на сервер с помощью *LoginRequest*, а сервер возвращает токен и информацию о пользователе в виде *AuthResponse*. *DTO*-классы помогают структурировать данные, упрощая сериализацию и десериализацию.

Вспомогательные классы, такие как *Constants*, содержат ключи и строки, которые часто используются в приложении, например, пути к *API* или ключи для локального хранилища. Это позволяет централизованно управлять такими значениями, упрощая поддержку и изменение кода.

Классы *Program* представляют собой точки входа в приложение и содержат настройку сервисов, *middleware* и инфраструктуры, необходимых для работы клиентской и серверной частей. Они определяют, как приложение будет инициализироваться, какие зависимости будут зарегистрированы и как будет происходить обработка запросов

Для клиентской части, реализованной в *Blazor WebAssembly*, класс *Program* отвечает за настройку *SPA*-приложения (*Single-Page Application*). Он создает экземпляр хоста с помощью *WebAssemblyHostBuilder*, подключает корневые компоненты (например, главный компонент приложения *App* и *MainLayout*) для управления метаинформацией страницы) и регистрирует сервисы, необходимые для обработки данных, аутентификации и взаимодействия с *API*. Здесь настраиваются такие ключевые элементы, как клиент *HTTP* для отправки запросов на сервер, локальное хранилище (*Blazored.LocalStorage*) для сохранения токенов, а также *AuthenticationStateProvider* для управления состоянием пользователя. Помимо этого, подключаются сторонние библиотеки, такие как *MudBlazor*, для создания пользовательского интерфейса.

Для серверной части, реализованной в *ASP.NET Core*, класс *Program* настраивает серверное приложение, ориентированное на предоставление *API*. Он конфигурирует подключение к базе данных через *Entity Framework Core*, добавляет поддержку системы управления пользователями на основе *Identity* и настраивает аутентификацию с использованием *JWT (JSON Web Token)*. Здесь происходит регистрация сервисов, таких как кэширование, обработка *HTTP*-запросов и уведомлений, а также добавление *Swagger* для документирования и тестирования *API*.

3 РЕЗУЛЬТАТЫ ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Описание пользовательского интерфейса

В качестве клиентского приложения реализовано веб-приложение с использованием фреймворка *Blazor WebAssembly (WASM)*. При запуске веб-приложения пользователь попадает на страницу авторизации. Пример страницы авторизации представлен на рисунке 3.1.

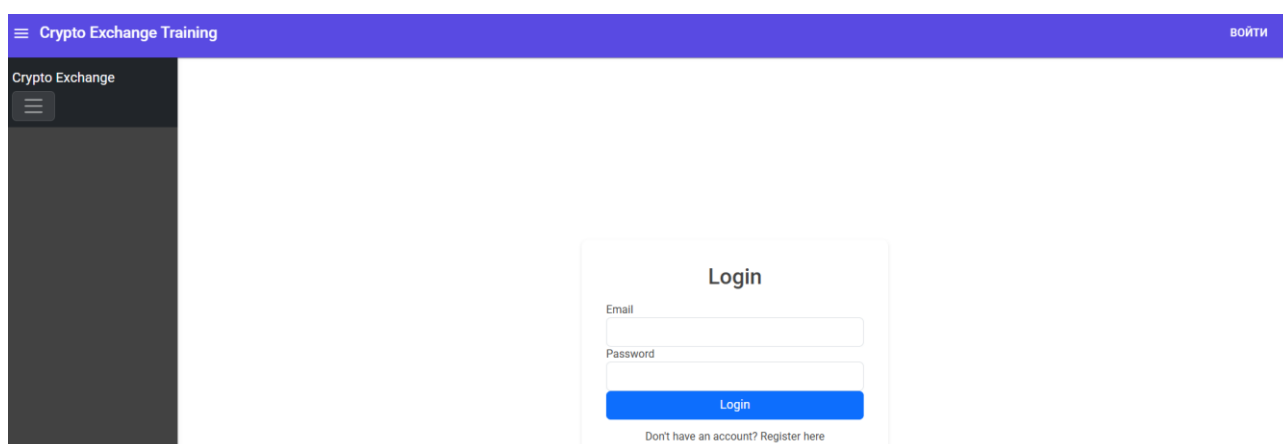
The screenshot shows the 'Login' page of the 'Crypto Exchange Training' application. The page has a purple header with the application name and a 'ВОЙТИ' (Login) button. A dark sidebar on the left contains the 'Crypto Exchange' logo and a menu icon. The main content area is white and features a 'Login' form. The form includes two input fields for 'Email' and 'Password', a blue 'Login' button, and a link that says 'Don't have an account? Register here'.

Рисунок 3.1 – Страница входа

Если же у пользователя нет аккаунта, то пользователь может перейти на страницу регистрации. Для этого пользователь должен нажать на ссылку «Нет аккаунта», после чего произойдет переход на страницу с регистрацией. Пример страницы регистрации представлен на рисунке 3.2.

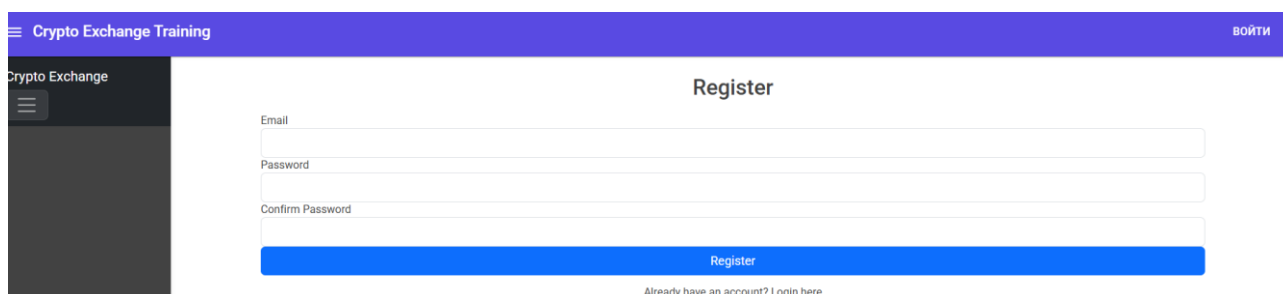
The screenshot shows the 'Register' page of the 'Crypto Exchange Training' application. The layout is consistent with the login page, featuring a purple header, a dark sidebar, and a white main content area. The 'Register' form includes three input fields for 'Email', 'Password', and 'Confirm Password', a blue 'Register' button, and a link that says 'Already have an account? Login here'.

Рисунок 3.2 – Страница регистрации

Чтобы зарегистрироваться, пользователь должен ввести свои данные, после того, как пользователь ввел свои данные, пользователь должен нажать на кнопку «Зарегистрироваться», после чего произойдет регистрация пользователя в системе. Результат регистрации пользователя представлен на рисунке 3.3.

Crypto Exchange Training

Профиль пользователя

Информация о пользователе

Email: user5@test.ru

Баланс: 10 000,00 Р

Дата регистрации: четверг, 16 января 2025 г. 06:36:32

Пополнение баланса

Сумма пополнения: 0

[Пополнить](#)

Рисунок 3.3 – Результат регистрации пользователя

Если пользователь неправильно ввел почту или пароль, то пользователь увидит сообщение об ошибке. Пример ошибки при регистрации представлен на рисунке 3.4.

Login

Invalid email or password

Email

likefortnite2019@mail.ru

Password

.....

[Login](#)

[Don't have an account? Register here](#)

Рисунок 3.4 – Ошибка при логине

Пользователь при желании может пополнить баланс нажав на кнопку «Профиль», после чего произойдет переход на страницу с профилем, где под

информацией о его профиле будет поля пополнения баланса. Пример данной страницы представлен на странице 3.5.

The screenshot shows a web application titled "Crypto Exchange Training". On the left is a dark sidebar with a "Crypto Exchange" header and a menu icon. The main content area is titled "Профиль пользователя" (User Profile). It contains two main sections: "Информация о пользователе" (User Information) and "Пополнение баланса" (Balance Replenishment). The "User Information" section has three input fields: "Email:" with the value "user2@test.ru", "Баланс:" (Balance) with the value "10 680,00 Р", and "Дата регистрации:" (Registration Date) with the value "среда, 15 января 2025 г. 02:32:49". The "Balance Replenishment" section has a "Сумма пополнения:" (Replenishment Sum) input field with the value "0" and a blue "Пополнить" (Replenish) button.

Рисунок 3.5 – Страница изменения профиля

Чтобы пополнить баланс, пользователю необходимо ввести в поле для пополнения баланса сумму пополнения и нажать кнопку «Пополнить». Пример результата представлен на рисунке 3.6.

This screenshot shows the same "Crypto Exchange Training" user profile page after a successful transaction. The "Баланс:" (Balance) field now displays "12 680,00 Р", indicating an increase from the previous state. The "Сумма пополнения:" (Replenishment Sum) field still shows "2000". The blue "Пополнить" (Replenish) button remains visible. A new green message box at the bottom of the "Пополнение баланса" section displays the text "Баланс успешно пополнен." (Balance successfully replenished).

Рисунок 3.6 – Пополняем баланс пользователя

На странице «Торговать» представлен интерфейс для торговли. Пример данной страницы представлен на рисунке 3.7.

Торговля

Торговая пара: Тип сделки:

Объем: Плечо:

Stop Loss: Take Profit:

ОТКРЫТЬ СДЕЛКУ

Активные сделки

ID	Пара	Тип	Объем	Цена входа	Stop Loss	Take Profit	Действия
13	ETHUSD	sell	1	3380,30000000	10	20	ЗАКРЫТЬ
12	BTCUSD	buy	0,1	99941,48000000	1	25	ЗАКРЫТЬ
11	BTCUSD	buy	0,1	99941,48000000	1	25	ЗАКРЫТЬ
10	BTCUSD	buy	1	99948,16000000	1	25	ЗАКРЫТЬ

Рисунок 3.7 – Страница торговли

Для того, чтобы открыть сделку пользователю необходимо выбрать торговую пару, после чего выбрать тип сделки и ввести объём, плечо, стоп-лосс и тейк-профит, после этого нажать кнопку «Открыть сделку». Пример результата открытия сделки представлен на рисунке 3.8.

Торговля

Торговая пара: Тип сделки:

Объем: Плечо:

Stop Loss: Take Profit:

ОТКРЫТЬ СДЕЛКУ

Активные сделки

ID	Пара	Тип	Объем	Цена входа	Stop Loss	Take Profit	Действия
14	BTCUSD	buy	100	99767,45000000	20	500	ЗАКРЫТЬ
13	ETHUSD	sell	1	3380,30000000	10	20	ЗАКРЫТЬ
12	BTCUSD	buy	0,1	99941,48000000	1	25	ЗАКРЫТЬ
11	BTCUSD	buy	0,1	99941,48000000	1	25	ЗАКРЫТЬ

Рисунок 3.8 – Результат открытия сделки

После создания сделки она начинает отображаться в списке активных сделок, где пользователь при желании может закрыть её. Пример закрытия сделки представлен на рисунке 3.9.

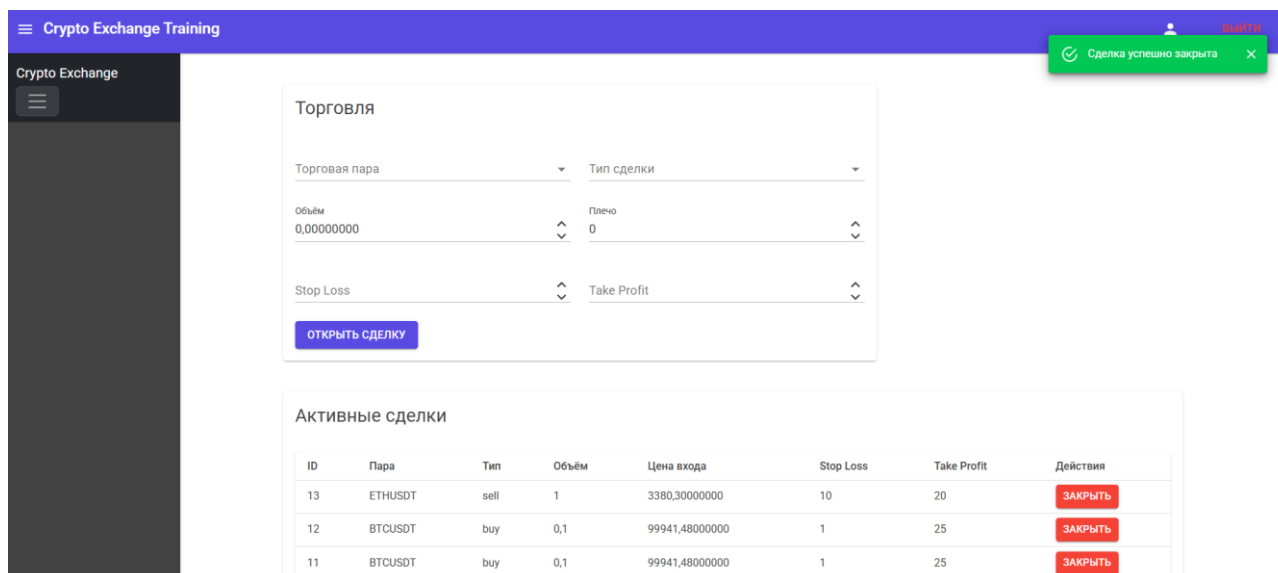


Рисунок 3.9 – Заккрытие сделки

Что посмотреть историю сделок, пользователю необходимо перейти на страницу «История сделок», где он может ознакомиться с историей своих сделок. Пример страницы представлен на рисунке 3.10.

Дата	Торговая пара	Тип	Объем	Цена входа	Статус
16.01.2025 07:14	BTCUSDT	buy	100	99767,45	open
16.01.2025 07:00	ETHUSDT	sell	1	3380,30	open
16.01.2025 01:18	BTCUSDT	buy	0,1	99941,48	open
16.01.2025 01:18	BTCUSDT	buy	0,1	99941,48	open
16.01.2025 01:18	BTCUSDT	buy	1	99948,16	open
16.01.2025 01:18	BTCUSDT	buy	1	99948,16	open
15.01.2025 23:34	BTCUSDT	buy	0,01	99986,74	open
15.01.2025 23:29	BTCUSDT	buy	0,01	100129,47	open

Рисунок 3.10 – История сделок

3.2 Верификация программного обеспечения

Для проверки разработанного приложения написаны модульные и нагрузочные тесты.

AuthControllerTests – тесты для проверки функционала аутентификации:

«*Register_Success*» – проверяет успешную регистрацию нового пользователя с корректными данными;

«*Login_Succes*» – проверяет успешную авторизацию существующего пользователя;

«*Register_Fails_WithExistingEmail*» – проверяет обработку ошибки при попытке регистрации с уже существующим *email*;

«*Login_Fails_WithWrongPassword*» – проверяет обработку ошибки при попытке входа с неверным паролем.

AssetControllerTests – тесты для проверки работы с активами пользователей:

«*GetUserAssets_Success*» – проверяет корректное получение списка активов авторизованного пользователя;

«*GetUserAssets_EmptyList_ForNewUser*» – проверяет получение пустого списка активов для нового пользователя;

«*Deposit_Success*» – проверяет успешное пополнение баланса актива;

«*Deposit_Fails_WithInvalidAmount*» – проверяет обработку ошибки при попытке пополнения на некорректную сумму.

TradeControllerTests – тесты для проверки торговых операций:

«*GetTradeHistory_Success*» – проверяет получение истории сделок пользователя;

«*GetTradeHistory_EmptyList*» – проверяет получение пустой истории сделок;

«*OpenTrade_Success*» – проверяет успешное открытие новой торговой позиции;

«*OpenTrade_Fails_InsufficientFunds*» – проверяет обработку ошибки при недостаточном балансе.

UserControllerTests – тесты для проверки операций с профилем пользователя:

«*GetProfile_Success*» – проверяет получение данных профиля авторизованного пользователя;

«*GetProfile_NotFound*» – проверяет обработку ошибки при запросе несуществующего профиля;

«*Deposit_Success*» – проверяет успешное пополнение баланса пользователя;

«*GetTransactionHistory_Success*» – проверяет получение истории транзакций.

MarketControllerTests – тесты для проверки рыночных данных:

«*GetMarketData_Success*» – проверяет получение актуальных рыночных данных;

«*GetOrderBook_Success*» – проверяет получение книги ордеров для выбранной торговой пары;

«*GetTradingPairs_Success*» – проверяет получение списка доступных торговых пар;

Результат запуска тестов представлен на рисунке 3.15.

```
Finished:    CryptoExchangeTests

Test Run Summary
Overall result: Passed
Test results: Total: 20, Passed: 20, Failed: 0, Skipped: 0
Start time: 11:11:54.168
End time: 11:11:56.337
Duration: 2.156 seconds

=== TEST EXECUTION SUMMARY ===
CryptoExchangeTests Total: 20, Passed: 20, Failed: 0, Skipped: 0
```

Рисунок 3.15 – Результаты модульных тестов

Таким образом, разработанное приложение прошло через модульные тесты и показало свою работоспособность.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано приложение «Учебная криптобиржа», предназначенное для обучения трейдингу криптовалютами в безопасной среде. Основная цель проекта заключалась в создании полноценной торговой платформы, реализованной с использованием современных веб-технологий и *WebSocket* для обеспечения *real-time* обновления данных.

В процессе работы над проектом были изучены принципы работы криптовалютных бирж, особенности маржинальной торговли и механизмы обработки торговых операций. Был разработан надежный механизм авторизации на основе JWT-токенов и система защиты данных с использованием *SSL/TLS* шифрования.

Разработанное программное обеспечение прошло серию модульных тестов с использованием *xUnit*, которые подтвердили корректность работы всех компонентов системы. Тесты продемонстрировали высокую производительность и точность обработки торговых операций.

Клиентская часть приложения реализована как одностраничное веб-приложение с использованием современного фреймворка *Blazor WebAssembly (WASM)*. Это обеспечивает удобный доступ к торговой платформе через любой браузер при наличии подключения к интернету. Интерфейс приложения разработан с учетом лучших практик *UX/UI* дизайна, что делает его интуитивно понятным для пользователей.

Функциональные возможности приложения включают создание и управление торговыми аккаунтами, выполнение торговых операций с использованием маржинального плеча, отслеживание текущих позиций и истории сделок, а также инструменты технического анализа для прогнозирования рыночных тенденций. Это позволяет пользователям получить полноценный опыт трейдинга без риска потери реальных средств.

Курсовой проект был выполнен полностью самостоятельно, с использованием современных технологий и подходов к разработке веб-приложений. Все компоненты системы, включая механизмы авторизации, обработки торговых операций и обновления данных в реальном времени, были реализованы без использования готовых решений или заимствования чужого кода.

Список использованных источников

1. Татенбаум Э. Компьютерные сети / Э. Татенбаум, Н. Фимстер, Д. Уэзеролл; 6-е изд. – СПб. : Питер, 2023. – 992 с.
2. Хабр. Сайт о программировании [Электронный ресурс] – Режим доступа: <https://habr.com/ru/articles/495698/> – Дата доступа: 08.12.2024.
3. METANIT. Руководство по ASP.NET Core [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/aspnetcore/> – Дата доступа: 05.12.2024.
4. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages / A. Freeman. – Apress, 2022. – 1054 p.
5. JWT Authentication & Authorization [Электронный ресурс] – Режим доступа: <https://jwt.io/introduction/> – Дата доступа: 07.12.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Руководство системного программиста

Для корректной работы приложения необходимо соблюдение следующих требований:

- поддерживаемые операционные системы *Windows 10* и выше;
- наличие следующих устройств ввода: стандартная клавиатура и компьютерная мышь;
- наличие устройства вывода – экран.

Программа состоит из двух составляющих – клиента и сервера. Для запуска как серверного, так и клиентского приложения необходимо развернуть клиентский сервер.

После необходимо зайти в корень проекта клиента и сервера и написать команду «*dotnet start*» для запуска проект, после чего произойдет запуск программы.

Пользователь может авторизовываться, создавать аккаунты, торговать, следить за сделками, пополнять баланс.

ПРИЛОЖЕНИЕ Б

(обязательное)

Руководство программиста

Для правильной работы приложения требуется соблюдение следующих требований:

- поддерживаемые операционные системы: *Windows 10* и *Windows 11*;
- для ввода данных необходимы стандартная клавиатура и компьютерная мышь;
- Для вывода информации требуется экран с минимальным разрешением 1280x720.

Для корректной работы программы также требуются следующие технические характеристики компьютера:

- минимальный объем оперативной памяти для установки и работы среды разработки *Microsoft Visual Studio* – 8 ГБ, рекомендуется 16 ГБ для комфортной работы;
- минимальные требования к процессору: архитектура x64, минимум *AMD Ryzen 3*, *Intel Core i3* 8-го поколения или аналогичный. Частота процессора должна составлять не менее 2,0 ГГц, рекомендуется 3,0 ГГц или выше;
- также рекомендуется наличие *SSD*-диска для ускорения работы системы и снижения времени компиляции.

Приложение работает в автоматическом режиме. Сначала необходимо запустить сервер, обрабатывающий запросы клиентов и работающий с базой данных. Затем необходимо запустить сервер, hostящий клиентское веб-приложение.

Для запуска как серверного, так и клиентского приложения необходимо развернуть клиентский сервер.

ПРИЛОЖЕНИЕ В

(обязательное)

Руководство пользователя

Руководство пользователя обеспечивает получение пользователем базовых навыков по эксплуатации данного приложения.

Разработанное приложения представляет собой учебную криптобиржу, на которой пользователь может регистрироваться, создавать сделки, следить за историей сделок, пополнять баланс.

Разработанное приложение имеет следующий функционал:

- регистрация и авторизация пользователя;
- создание сделок для торговли, пополнение баланса;

Для использования программного приложения пользователь должен быть ознакомлен с настоящим руководством пользователя.

Приложения предназначено для пользователей, который обучиться торговле криптоактивами.

Для корректной работы приложения необходимо соблюдение следующих требований:

- поддерживаемые операционные системы *Windows 10* и выше;
- современный браузер с поддержкой *ES2016*, например, *OperaGX*, *Yandex Browser*, *Google Chrome*;
- устройства ввода: мышь и клавиатура;
- хорошее подключение к интернету.

Для работы необходимо запустить браузер.

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования, описанные в пункте 3 данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова.

ПРИЛОЖЕНИЕ Г

(обязательное)

Листинг программы

```
using System.Reflection;
using System.Text;
using CryptoExchangeTrainingAPI.Data;
using CryptoExchangeTrainingAPI.Models;
using CryptoExchangeTrainingAPI.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Swashbuckle.AspNetCore.SwaggerGen;

var builder = WebApplication.CreateBuilder(args);

// Добавление Swagger (OpenAPI)
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory,
    $"{ Assembly.GetExecutingAssembly().GetName().Name }.xml"));
});

// Настройка подключения к базе данных
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection"))
    .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking)); // Оптимизация запросов

// Настройка Identity
builder.Services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequiredLength = 8;
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5); // Блокировка после
неудачных попыток входа
    options.Lockout.MaxFailedAccessAttempts = 5; // Максимальное количество попыток
    options.User.RequireUniqueEmail = true; // Требование уникального email
});

// Настройка JWT
var jwtSettings = builder.Configuration.GetSection("JwtSettings");
```

```

var key = Encoding.UTF8.GetBytes(jwtSettings["Secret"]);
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = jwtSettings["Issuer"],
        ValidAudience = jwtSettings["Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ClockSkew = TimeSpan.Zero // Убираем дополнительное время жизни токена
    };
});

// Регистрация сервисов
builder.Services.AddHttpClient<IHttpRequestService, HttpRequestService>();
builder.Services.AddSingleton<ICacheService, CacheService>();
builder.Services.AddScoped<IMarketService, MarketService>();
builder.Services.AddScoped<INotificationService, NotificationService>();
builder.Services.AddScoped<JwtTokenService>();

// Добавление контроллеров
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.ReferenceHandler =
        System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
        options.JsonSerializerOptions.DefaultIgnoreCondition =
        System.Text.Json.Serialization.JsonIgnoreCondition.WhenWritingNull;
        options.JsonSerializerOptions.PropertyNamingPolicy =
        System.Text.Json.JsonNamingPolicy.CamelCase; // Используем camelCase
        options.JsonSerializerOptions.PropertyNameCaseInsensitive = true;
    });

// Настройка CORS
// Добавление CORS
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowSpecificOrigins",
        policy =>
        {
            policy.WithOrigins("https://localhost:7160") // Укажите адрес вашего Blazor клиента
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials(); // Если используются куки или заголовки аутентификации
        });
});

```

```

    });
});
var app = builder.Build();
// Использование Swagger для тестирования API
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
else
{
    app.UseHttpsRedirection(); // Перенаправление HTTP на HTTPS
}
// Использование CORS
app.UseCors("AllowSpecificOrigins");
// Использование аутентификации и авторизации
app.UseAuthentication();
app.UseAuthorization();
// Добавление WebSocket middleware
app.UseWebSockets();
// Маршруты для контроллеров
app.MapControllers();
// Запуск приложения
app.Run();

using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Identity;

namespace CryptoExchangeTrainingAPI.Models
{
    public class User : IdentityUser
    {
        public decimal Balance { get; set; } = 10000.00m; // Начальный баланс пользователя
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow; // Время регистрации
        public DateTime? LastLoginAt { get; set; } // Время последнего входа
        public ICollection<Trade> Trades { get; set; } = new List<Trade>();
        public ICollection<UserAsset> UserAssets { get; set; } = new List<UserAsset>();
        public ICollection<Notification> Notifications { get; set; } = new List<Notification>();
    }
}

namespace CryptoExchangeTrainingAPI.Models
{
    public class TradeDto
    {
        public int Id { get; set; }
        public string Pair { get; set; } = null!;
        public string Type { get; set; } = null!;
        public int Leverage { get; set; }
        public decimal Amount { get; set; }
        public decimal EntryPrice { get; set; }
    }
}

```



```

        public decimal? StopLoss { get; set; }
        public decimal? TakeProfit { get; set; }
        public string Status { get; set; } = "open";
        public DateTime OpenedAt { get; set; }
        public DateTime? ClosedAt { get; set; }
    }
}
using System;
using CryptoExchangeTrainingAPI.Models;

using System.Text.Json.Serialization;

namespace CryptoExchangeTrainingAPI.Models
{
    public class Trade
    {
        public int Id { get; set; } // Уникальный идентификатор сделки
        public string UserId { get; set; } = null!; // Внешний ключ на пользователя
        public string Pair { get; set; } = null!; // Торговая пара (например, BTC/USDT)
        public string Type { get; set; } = null!; // Тип сделки ("buy" или "sell")
        public int Leverage { get; set; } // Маржинальное плечо
        public decimal Amount { get; set; } // Объем сделки (например, 0.01 BTC)
        public decimal EntryPrice { get; set; } // Цена открытия сделки
        public decimal? ExitPrice { get; set; } // Цена закрытия сделки (если сделка закрыта)
        public decimal? StopLoss { get; set; } // Уровень Stop Loss
        public decimal? TakeProfit { get; set; } // Уровень Take Profit
        public decimal? ProfitLoss { get; set; } // Прибыль/убыток сделки
        public string Status { get; set; } = "open"; // Статус сделки ("open" или "closed")
        public DateTime OpenedAt { get; set; } = DateTime.UtcNow; // Время открытия сделки
        public DateTime? ClosedAt { get; set; } // Время закрытия сделки
        public decimal? Fee { get; set; }

        [JsonIgnore]
        public User? User { get; set; }
    }
}
namespace CryptoExchangeTrainingAPI.Models
{
    public class Token
    {
        public int Id { get; set; } // Уникальный идентификатор токена
        public string Name { get; set; } = string.Empty; // Название токена (например, BTC, ETH)
        public string Symbol { get; set; } = string.Empty; // Символ токена (например, BTC, ETH)

        public ICollection<UserAsset> UserAssets { get; set; } = new List<UserAsset>(); // Связь с активами пользователей
    }
}
namespace CryptoExchangeTrainingAPI.Models
{

```

```

public class RegisterDto
{
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
}

using Newtonsoft.Json;
using System.Collections.Generic;

namespace CryptoExchangeTrainingAPI.Models
{
    public class OrderBookResponse
    {
        [JsonProperty("lastUpdateId")]
        public long LastUpdateId { get; set; }

        [JsonProperty("bids")]
        public List<List<string>> Bids { get; set; } = null!;

        [JsonProperty("asks")]
        public List<List<string>> Asks { get; set; } = null!;
    }
}
using System;
using CryptoExchangeTrainingAPI.Models;

namespace CryptoExchangeTrainingAPI.Models
{
    public class Order
    {
        public int Id { get; set; }
        public string UserId { get; set; } = null!; // Внешний ключ на пользователя
        public string Pair { get; set; } = null!;
        public string Type { get; set; } = null!;
        public decimal Amount { get; set; }
        public decimal Price { get; set; }
        public string Status { get; set; } = "pending";
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        // Связь с пользователем
        public User? User { get; set; }
    }
}

namespace CryptoExchangeTrainingAPI.Models
{
    public class OpenTradeRequestDto
    {
        public string Pair { get; set; } = null!; // Торговая пара
        public string Type { get; set; } = null!; // Тип сделки ("buy" или "sell")
        public int Leverage { get; set; } // Маржинальное плечо
    }
}

```

```

        public decimal Amount { get; set; } // Объем сделки
        public decimal? StopLoss { get; set; } // Уровень Stop Loss
        public decimal? TakeProfit { get; set; } // Уровень Take Profit
    }
}
using System;

namespace CryptoExchangeTrainingAPI.Models
{
    public class Notification
    {
        public int Id { get; set; } // Уникальный идентификатор уведомления
        public string UserId { get; set; } = null!; // Внешний ключ на пользователя
        public string Message { get; set; } = null!; // Текст уведомления
        public bool IsRead { get; set; } = false; // Прочитано или нет
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow; // Время создания уведомления

        // Связь с пользователем
        public User? User { get; set; }
    }
}
using System;

namespace CryptoExchangeTrainingAPI.Models
{
    public class MarketData
    {
        public int Id { get; set; } // Уникальный идентификатор записи
        public string Pair { get; set; } = null!; // Торговая пара
        public decimal Price { get; set; } // Цена криптовалюты
        public DateTime Timestamp { get; set; } = DateTime.UtcNow; // Время записи
    }
}
namespace CryptoExchangeTrainingAPI.Models
{
    public class LoginDto
    {
        public string Email { get; set; } = null!;
        public string Password { get; set; } = null!;
    }
}
using System;

namespace CryptoExchangeTrainingAPI.Models
{
    public class HistoricalData
    {
        public int Id { get; set; }
        public string Pair { get; set; } = null!;
        public DateTime Timestamp { get; set; }
    }
}

```

```

        public decimal Price { get; set; }
    }
}
using System.ComponentModel.DataAnnotations;

namespace CryptoExchangeTrainingAPI.Models
{
    public class DepositRequestDto
    {
        [Range(0.01, double.MaxValue, ErrorMessage = "Сумма должна быть больше 0.")]
        public decimal Amount { get; set; }
        public string Asset { get; set; } = string.Empty;
    }
}

namespace CryptoExchangeTrainingAPI.Models.Response
{
    public class ProfileResponse
    {
        public string Email { get; set; }
        public decimal Balance { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime? LastLoginAt { get; set; } // добавляем ?
        public List<AssetResponse> Assets { get; set; }
    }
}

namespace CryptoExchangeTrainingAPI.Models.Response
{
    public class AssetResponse
    {
        public string Symbol { get; set; }
        public decimal Quantity { get; set; }
        public decimal CurrentPrice { get; set; }
    }
}
// <auto-generated />
using System;
using CryptoExchangeTrainingAPI.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace CryptoExchangeTrainingAPI.Migrations
{
    [DbContext(typeof(ApplicationDbContext))]
    partial class ApplicationDbContextModelSnapshot : ModelSnapshot
    {

```

```

protected override void BuildModel(ModelBuilder modelBuilder)
{
#pragma warning disable 612, 618
    modelBuilder
        .HasAnnotation("ProductVersion", "9.0.0")
        .HasAnnotation("Relational:MaxIdentifierLength", 63);

    NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

    modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.HistoricalData", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("integer");

        NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

        b.Property<string>("Pair")
            .IsRequired()
            .HasColumnType("text");

        b.Property<decimal>("Price")
            .HasColumnType("numeric");

        b.Property<DateTime>("Timestamp")
            .HasColumnType("timestamp with time zone");

        b.HasKey("Id");

        b.ToTable("HistoricalData");
    });

    modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.MarketData", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("integer");

        NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

        b.Property<string>("Pair")
            .IsRequired()
            .HasColumnType("text");

        b.Property<decimal>("Price")
            .HasColumnType("numeric");

        b.Property<DateTime>("Timestamp")
            .HasColumnType("timestamp with time zone");

        b.HasKey("Id");
    });

```

```

        b.ToTable("MarketData");
    });

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Notification", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<DateTime>("CreatedAt")
        .HasColumnType("timestamp with time zone");

    b.Property<bool>("IsRead")
        .HasColumnType("boolean");

    b.Property<string>("Message")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("UserId")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

    b.HasIndex("UserId");

    b.ToTable("Notifications");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Order", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<decimal>("Amount")
        .HasColumnType("numeric");

    b.Property<DateTime>("CreatedAt")
        .HasColumnType("timestamp with time zone");

    b.Property<string>("Pair")
        .IsRequired()
        .HasColumnType("text");

```

```

        b.Property<decimal>("Price")
            .HasColumnType("numeric");

        b.Property<string>("Status")
            .IsRequired()
            .HasColumnType("text");

        b.Property<string>("Type")
            .IsRequired()
            .HasColumnType("text");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");

        b.HasIndex("UserId");

        b.ToTable("Orders");
    });

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Token", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<string>("Name")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("Symbol")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

    b.ToTable("Tokens");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Trade", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

```

```

b.Property<decimal>("Amount")
    .HasColumnType("numeric");

b.Property<DateTime?>("ClosedAt")
    .HasColumnType("timestamp with time zone");

b.Property<decimal>("EntryPrice")
    .HasColumnType("numeric");

b.Property<decimal?>("ExitPrice")
    .HasColumnType("numeric");

b.Property<decimal?>("Fee")
    .HasColumnType("numeric");

b.Property<int>("Leverage")
    .HasColumnType("integer");

b.Property<DateTime>("OpenedAt")
    .HasColumnType("timestamp with time zone");

b.Property<string>("Pair")
    .IsRequired()
    .HasColumnType("text");

b.Property<decimal?>("ProfitLoss")
    .HasColumnType("numeric");

b.Property<string>("Status")
    .IsRequired()
    .HasColumnType("text");

b.Property<decimal?>("StopLoss")
    .HasColumnType("numeric");

b.Property<decimal?>("TakeProfit")
    .HasColumnType("numeric");

b.Property<string>("Type")
    .IsRequired()
    .HasColumnType("text");

b.Property<string>("UserId")
    .IsRequired()
    .HasColumnType("text");

b.HasKey("Id");

b.HasIndex("UserId");

b.ToTable("Trades");

```



```

});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.User", b =>
{
    b.Property<string>("Id")
        .HasColumnType("text");

    b.Property<int>("AccessFailedCount")
        .HasColumnType("integer");

    b.Property<decimal>("Balance")
        .HasColumnType("numeric");

    b.Property<string>("ConcurrencyStamp")
        .IsConcurrencyToken()
        .HasColumnType("text");

    b.Property<DateTime>("CreatedAt")
        .HasColumnType("timestamp with time zone");

    b.Property<string>("Email")
        .HasMaxLength(256)
        .HasColumnType("character varying(256)");

    b.Property<bool>("EmailConfirmed")
        .HasColumnType("boolean");

    b.Property<DateTime?>("LastLoginAt")
        .HasColumnType("timestamp with time zone");

    b.Property<bool>("LockoutEnabled")
        .HasColumnType("boolean");

    b.Property<DateTimeOffset?>("LockoutEnd")
        .HasColumnType("timestamp with time zone");

    b.Property<string>("NormalizedEmail")
        .HasMaxLength(256)
        .HasColumnType("character varying(256)");

    b.Property<string>("NormalizedUserName")
        .HasMaxLength(256)
        .HasColumnType("character varying(256)");

    b.Property<string>("PasswordHash")
        .HasColumnType("text");

    b.Property<string>("PhoneNumber")
        .HasColumnType("text");

    b.Property<bool>("PhoneNumberConfirmed")

```

```

        .HasColumnType("boolean");

b.Property<string>("SecurityStamp")
    .HasColumnType("text");

b.Property<bool>("TwoFactorEnabled")
    .HasColumnType("boolean");

b.Property<string>("UserName")
    .HasMaxLength(256)
    .HasColumnType("character varying(256)");

b.HasKey("Id");

b.HasIndex("NormalizedEmail")
    .HasDatabaseName("EmailIndex");

b.HasIndex("NormalizedUserName")
    .IsUnique()
    .HasDatabaseName("UserNameIndex");

b.ToTable("AspNetUsers", (string)null);
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.UserAsset", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<string>("Asset")
        .IsRequired()
        .HasColumnType("text");

    b.Property<decimal>("Balance")
        .HasColumnType("numeric");

    b.Property<int?>("TokenId")
        .HasColumnType("integer");

    b.Property<DateTime>("UpdatedAt")
        .HasColumnType("timestamp with time zone");

    b.Property<string>("UserId")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

```

```

        b.HasIndex("TokenId");

        b.HasIndex("UserId");

        b.ToTable("UserAssets");
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole", b =>
{
    b.Property<string>("Id")
        .HasColumnType("text");

    b.Property<string>("ConcurrencyStamp")
        .IsConcurrencyToken()
        .HasColumnType("text");

    b.Property<string>("Name")
        .HasMaxLength(256)
        .HasColumnType("character varying(256)");

    b.Property<string>("NormalizedName")
        .HasMaxLength(256)
        .HasColumnType("character varying(256)");

    b.HasKey("Id");

    b.HasIndex("NormalizedName")
        .IsUnique()
        .HasDatabaseName("RoleNameIndex");

    b.ToTable("AspNetRoles", (string)null);
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<string>("ClaimType")
        .HasColumnType("text");

    b.Property<string>("ClaimValue")
        .HasColumnType("text");

    b.Property<string>("RoleId")
        .IsRequired()
        .HasColumnType("text");

```

```

        b.HasKey("Id");

        b.HasIndex("RoleId");

        b.ToTable("AspNetRoleClaims", (string)null);
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("integer");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

    b.Property<string>("ClaimType")
        .HasColumnType("text");

    b.Property<string>("ClaimValue")
        .HasColumnType("text");

    b.Property<string>("UserId")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

    b.HasIndex("UserId");

    b.ToTable("AspNetUserClaims", (string)null);
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>", b =>
{
    b.Property<string>("LoginProvider")
        .HasColumnType("text");

    b.Property<string>("ProviderKey")
        .HasColumnType("text");

    b.Property<string>("ProviderDisplayName")
        .HasColumnType("text");

    b.Property<string>("UserId")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("LoginProvider", "ProviderKey");

    b.HasIndex("UserId");

```

```

        b.ToTable("AspNetUserLogins", (string)null);
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>", b =>
{
    b.Property<string>("UserId")
        .HasColumnType("text");

    b.Property<string>("RoleId")
        .HasColumnType("text");

    b.HasKey("UserId", "RoleId");

    b.HasIndex("RoleId");

    b.ToTable("AspNetUserRoles", (string)null);
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>", b =>
{
    b.Property<string>("UserId")
        .HasColumnType("text");

    b.Property<string>("LoginProvider")
        .HasColumnType("text");

    b.Property<string>("Name")
        .HasColumnType("text");

    b.Property<string>("Value")
        .HasColumnType("text");

    b.HasKey("UserId", "LoginProvider", "Name");

    b.ToTable("AspNetUserTokens", (string)null);
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Notification", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", "User")
        .WithMany("Notifications")
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.Navigation("User");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Order", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", "User")

```

```

        .WithMany()
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.Navigation("User");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Trade", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", "User")
        .WithMany("Trades")
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.Navigation("User");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.UserAsset", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.Token", "Token")
        .WithMany("UserAssets")
        .HasForeignKey("TokenId");

    b.HasOne("CryptoExchangeTrainingAPI.Models.User", "User")
        .WithMany("UserAssets")
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.Navigation("Token");

    b.Navigation("User");
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b =>
{
    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
        .WithMany()
        .HasForeignKey("RoleId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", null)
        .WithMany()
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)

```

```

        .IsRequired();
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", null)
        .WithMany()
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>", b =>
{
    b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
        .WithMany()
        .HasForeignKey("RoleId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.HasOne("CryptoExchangeTrainingAPI.Models.User", null)
        .WithMany()
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>", b =>
{
    b.HasOne("CryptoExchangeTrainingAPI.Models.User", null)
        .WithMany()
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.Token", b =>
{
    b.Navigation("UserAssets");
});

modelBuilder.Entity("CryptoExchangeTrainingAPI.Models.User", b =>
{
    b.Navigation("Notifications");

    b.Navigation("Trades");

    b.Navigation("UserAssets");
});
#pragma warning restore 612, 618
}

```

```

    }
}
using Microsoft.EntityFrameworkCore;
using CryptoExchangeTrainingAPI.Models;
using System.Collections.Generic;
using System.Reflection.Emit;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace CryptoExchangeTrainingAPI.Data
{
    public class ApplicationDbContext : IdentityDbContext<User>
    {
        // Конструктор для передачи параметров конфигурации
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

        // DbSet для каждой таблицы
        public DbSet<Trade> Trades { get; set; } = null!;
        public DbSet<MarketData> MarketData { get; set; } = null!;
        public DbSet<UserAsset> UserAssets { get; set; } = null!;
        public DbSet<Notification> Notifications { get; set; } = null!;
        public DbSet<HistoricalData> HistoricalData { get; set; } = null!;
        public DbSet<Order> Orders { get; set; } = null!;
        public DbSet<Token> Tokens { get; set; } = null!;

        public new Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)
        {
            return base.SaveChangesAsync(cancellationToken);
        }

        // Настройка моделей с помощью Fluent API
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<User>()
                .HasMany(u => u.Trades)
                .WithOne(t => t.User)
                .HasForeignKey(t => t.UserId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<User>()
                .HasMany(u => u.UserAssets)
                .WithOne(ua => ua.User)
                .HasForeignKey(ua => ua.UserId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<User>()
                .HasMany(u => u.Notifications)
                .WithOne(n => n.User)
                .HasForeignKey(n => n.UserId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
}

```



```

    }
    }
}
using Microsoft.AspNetCore.Mvc;
using System.Net.WebSockets;
using System.Text;
using CryptoExchangeTrainingAPI.Services;

namespace CryptoExchangeTrainingAPI.Controllers
{
    [ApiController]
    [Route("ws")]
    public class WebSocketController : ControllerBase
    {
        private readonly IMarketService _marketService;
        private readonly ILogger<WebSocketController> _logger;
        public WebSocketController(IMarketService marketService, ILogger<WebSocketController> logger)
        {
            _marketService = marketService;
            _logger = logger;
        }

        [HttpGet("price")]
        public async Task GetPriceWebSocket()
        {
            if (HttpContext.WebSockets.IsWebSocketRequest)
            {
                using var webSocket = await HttpContext.WebSockets.AcceptWebSocketAsync();
                _logger.LogInformation("WebSocket connection opened.");
                await Echo(webSocket);
            }
            else
            {
                HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;
            }
        }

        private async Task Echo(WebSocket webSocket)
        {
            var buffer = new byte[1024 * 4];
            WebSocketReceiveResult result;

            try
            {
                do
                {
                    result = await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationTokens.None);
                    if (result.MessageType == WebSocketMessageType.Text)
                    {
                        var message = Encoding.UTF8.GetString(buffer, 0, result.Count);

```

```

        var pair = message.ToUpper();
        _logger.LogInformation("Received pair message: {pair}", pair);
        while (webSocket.State == WebSocketState.Open)
        {
            try
            {
                var price = await _marketService.GetPriceAsync(pair);
                var response = Encoding.UTF8.GetBytes($"{{\"pair\": \"{pair}\", \"price\": 
\\\"{price}\\\"}}});
                await webSocket.SendAsync(new ArraySegment<byte>(response, 0, response.Length),
                WebSocketMessageType.Text, true, CancellationToken.None);
                await Task.Delay(3000); // отправка цены раз в 3 секунды
            }
            catch (Exception e)
            {
                _logger.LogError(e, "Error send data via websocket");
                break;
            }
        }
    } while (!result.CloseStatus.HasValue);
}
catch (Exception e)
{
    _logger.LogError(e, "Error in websocket connection");
}
finally
{
    _logger.LogInformation("WebSocket connection closed.");
    await webSocket.CloseAsync(WebSocketCloseStatus.NormalClosure, "Connection closed",
    CancellationToken.None);
}

}
}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Data;
using Microsoft.AspNetCore.Authorization;
using CryptoExchangeTrainingAPI.Models;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace CryptoExchangeTrainingAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class UserController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

```

```

public UserController(ApplicationDbContext context)
{
    _context = context;
}

[HttpGet("profile")]
[Authorize]
public async Task<IActionResult> GetProfile()
{
    var userId = User.FindFirst("id")?.Value;
    if (userId == null)
    {
        return Unauthorized("Пользователь не найден.");
    }
    var user = await _context.Users.FindAsync(userId);

    if (user == null)
    {
        return Unauthorized("Пользователь не найден.");
    }

    var assets = await _context.UserAssets
        .Where(a => a.UserId == userId)
        .Select(a => new
        {
            a.Asset,
            a.Balance
        })
        .ToListAsync();

    var assetsWithPrice = new List<object>();

    foreach (var asset in assets)
    {
        // Временно установим цену в 100, тут будет логика получения реальной цены
        decimal currentPrice = 100.00m;

        assetsWithPrice.Add(new
        {
            Symbol = asset.Asset,
            Quantity = asset.Balance,
            CurrentPrice = currentPrice
        });
    }

    return Ok(new
    {
        user.Email,
        user.Balance,

```

```

        user.CreatedAt,
        user.LastLoginAt,
        Assets = assetsWithPrice
    });
}

[HttpPost("deposit")]
[Authorize]
public async Task<IActionResult> Deposit([FromBody] DepositRequestDto request)
{
    if (request == null || request.Amount <= 0)
    {
        return BadRequest("Сумма пополнения должна быть больше нуля.");
    }

    var userId = User.FindFirst("id")?.Value;
    var user = await _context.Users.FindAsync(userId);

    if (user == null)
    {
        return Unauthorized("Пользователь не найден.");
    }

    // Логирование перед изменением
    Console.WriteLine($"Баланс до пополнения: {user.Balance}");

    user.Balance += request.Amount;
    await _context.SaveChangesAsync();

    // Логирование после сохранения изменений
    Console.WriteLine($"Баланс после пополнения: {user.Balance}");

    return Ok(new { Balance = user.Balance });
}

[HttpGet("transactions")]
[Authorize]
public async Task<IActionResult> GetTransactionHistory()
{
    var userId = User.FindFirst("id")?.Value;

    // Проверяем, что пользователь авторизован
    if (userId == null)
    {
        return Unauthorized(new { Success = false, Message = "Пользователь не авторизован." });
    }

    // Получаем сделки пользователя из базы данных
    var trades = await _context.Trades
        .Where(t => t.UserId == userId)
        .OrderByDescending(t => t.OpenedAt)

```

```

        .ToListAsync();

// Преобразуем сделку в DTO, если нужно
var tradeDtos = trades.Select(t => new TradeDto
{
    Id = t.Id,
    Pair = t.Pair,
    Type = t.Type,
    Leverage = t.Leverage,
    Amount = t.Amount,
    EntryPrice = t.EntryPrice,
    StopLoss = t.StopLoss,
    TakeProfit = t.TakeProfit,
    Status = t.Status,
    OpenedAt = t.OpenedAt,
    ClosedAt = t.ClosedAt
}).ToList();

return Ok(new
{
    Success = true,
    Data = tradeDtos
});
}

}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Data;
using Microsoft.AspNetCore.Authorization;
using CryptoExchangeTrainingAPI.Models;
using Microsoft.EntityFrameworkCore;
using CryptoExchangeTrainingAPI.Services;
using System.Globalization;

namespace CryptoExchangeTrainingAPI.Controllers
{
    /// <summary>
    /// Контроллер для управления сделками.
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class TradeController : ControllerBase
    {
        private readonly ApplicationDbContext _context;
        private readonly IMarketService _marketService;
        private readonly INotificationService _notificationService;

        /// <summary>
        /// Конструктор контроллера TradeController.

```

```

/// </summary>
/// <param name="context">Контекст базы данных.</param>
/// <param name="marketService">Сервис для получения рыночных данных.</param>
/// <param name="notificationService">Сервис для отправки уведомлений.</param>
public TradeController(ApplicationDbContext context, IMarketService marketService,
INotificationService notificationService)
{
    _context = context;
    _marketService = marketService;
    _notificationService = notificationService;
}

/// <summary>
/// Получить историю сделок текущего пользователя.
/// </summary>
/// <returns>Список сделок.</returns>
[HttpGet("history")]
[Authorize]
public async Task<IActionResult> GetTradeHistory()
{
    var userId = User.FindFirst("id")?.Value;

    var trades = await _context.Trades
        .Where(t => t.UserId == userId)
        .OrderByDescending(t => t.OpenedAt)
        .Select(t => new TradeDto
        {
            Id = t.Id,
            Pair = t.Pair,
            Type = t.Type,
            Leverage = t.Leverage,
            Amount = t.Amount,
            EntryPrice = t.EntryPrice,
            StopLoss = t.StopLoss,
            TakeProfit = t.TakeProfit,
            Status = t.Status,
            OpenedAt = t.OpenedAt,
            ClosedAt = t.ClosedAt
        })
        .ToListAsync();

    return Ok(trades);
}

/// <summary>
/// Открыть новую сделку.
/// </summary>
/// <param name="request">Данные для открытия сделки.</param>
/// <returns>Информация о созданной сделке.</returns>
[HttpPost("open")]
[Authorize]

```

```

public async Task<IActionResult> OpenTrade([FromBody] OpenTradeRequestDto request)
{
    if (request == null || string.IsNullOrEmpty(request.Pair) || string.IsNullOrEmpty(request.Type) ||
request.Amount <= 0)
    {
        return BadRequest("Некорректные данные для открытия сделки.");
    }

    var userId = User.FindFirst("id")?.Value;
    var user = await _context.Users.FindAsync(userId);

    if (user == null)
    {
        return Unauthorized("Пользователь не найден.");
    }

    var fee = request.Amount * 0.001m; // Пример комиссии

    if (user.Balance < request.Amount + fee)
    {
        return BadRequest("Недостаточно средств для открытия сделки с учетом комиссии.");
    }

    user.Balance -= request.Amount + fee;

    var orderBook = await _marketService.GetOrderBookAsync(request.Pair);
    if (orderBook == null || orderBook.Asks == null || orderBook.Bids == null || (!orderBook.Asks.Any()
&& !orderBook.Bids.Any()))
    {
        return BadRequest("Не удалось получить данные о рыночных ордерах.");
    }

    decimal price;
    if (request.Type.ToLower() == "buy")
    {
        price = decimal.Parse(orderBook.Asks.First()[0], CultureInfo.InvariantCulture);
    }
    else
    {
        price = decimal.Parse(orderBook.Bids.First()[0], CultureInfo.InvariantCulture);
    }

    // Создание сделки
    var trade = new Trade
    {
        UserId = userId,
        Pair = request.Pair.ToUpper(),
        Type = request.Type.ToLower(),
        Leverage = request.Leverage,
        Amount = request.Amount,
        EntryPrice = price, // использовать цену из книги ордеров
    }
}

```

```

        StopLoss = request.StopLoss,
        TakeProfit = request.TakeProfit,
        OpenedAt = DateTime.UtcNow,
        Status = "open",
        Fee = fee,
    };

    _context.Trades.Add(trade);
    await _context.SaveChangesAsync();
    await _notificationService.CreateNotificationAsync(userId, $"Сделка на {request.Pair} успешно
открыта.");

    // Возвращаем TradeDto
    var tradeDto = new TradeDto
    {
        Id = trade.Id,
        Pair = trade.Pair,
        Type = trade.Type,
        Leverage = trade.Leverage,
        Amount = trade.Amount,
        EntryPrice = trade.EntryPrice,
        StopLoss = trade.StopLoss,
        TakeProfit = trade.TakeProfit,
        Status = trade.Status,
        OpenedAt = trade.OpenedAt,
        ClosedAt = trade.ClosedAt
    };

    return Ok(tradeDto);
}
/// <summary>
/// Закрыть сделку.
/// </summary>
/// <param name="id">ID сделки.</param>
/// <returns>Информация о закрытой сделке.</returns>
[HttpPost("close/{id}")]
[Authorize]
public async Task<IActionResult> CloseTrade(int id)
{
    var userId = User.FindFirst("id")?.Value;
    var trade = await _context.Trades.FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);

    if (trade == null)
    {
        return NotFound("Сделка не найдена.");
    }

    if (trade.Status != "open")
    {
        return BadRequest("Сделка уже закрыта.");
    }
}

```



```

// Получение текущей цены
var exitPrice = await _marketService.GetPriceAsync(trade.Pair);
trade.ExitPrice = exitPrice;

// Расчет прибыли/убытка
var profitLoss = (exitPrice - trade.EntryPrice) * trade.Amount * trade.Leverage;
trade.ProfitLoss = profitLoss;

// Обновление баланса пользователя
var user = await _context.Users.FindAsync(userId);
if (user != null)
{
    user.Balance += profitLoss;

    // Логика работы с активами
    if (trade.Type.ToLower() == "buy")
    {
        var userAsset = await _context.UserAssets.FirstOrDefaultAsync(ua => ua.UserId == userId &&
ua.Asset == trade.Pair);
        if (userAsset == null)
        {
            await _context.UserAssets.AddAsync(new UserAsset { UserId = userId, Asset = trade.Pair,
Balance = trade.Amount });
        }
        else
        {
            userAsset.Balance += trade.Amount;
        }
        await _context.SaveChangesAsync();
    }
    else if (trade.Type.ToLower() == "sell")
    {
        var userAsset = await _context.UserAssets.FirstOrDefaultAsync(ua => ua.UserId == userId &&
ua.Asset == trade.Pair);
        if (userAsset != null)
        {
            userAsset.Balance -= trade.Amount;
        }

        await _context.SaveChangesAsync();
    }
}

// Закрытие сделки
trade.Status = "closed";
trade.ClosedAt = DateTime.UtcNow;
await _context.SaveChangesAsync();
await _notificationService.CreateNotificationAsync(userId, $"Сделка на {trade.Pair} успешно
закрыта с прибылью {profitLoss}");

```

```

        return Ok(trade);
    }

}

}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Data;
using Microsoft.AspNetCore.Authorization;
using CryptoExchangeTrainingAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace CryptoExchangeTrainingAPI.Controllers
{
    /// <summary>
    /// Контроллер для управления уведомлениями.
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class NotificationController : ControllerBase
    {
        private readonly ApplicationDbContext _context;
        /// <summary>
        /// Конструктор контроллера NotificationController.
        /// </summary>
        /// <param name="context">Контекст базы данных.</param>
        public NotificationController(ApplicationDbContext context)
        {
            _context = context;
        }

        /// <summary>
        /// Получить список уведомлений текущего пользователя.
        /// </summary>
        /// <returns>Список уведомлений пользователя.</returns>
        [HttpGet]
        [Authorize]
        public async Task<IActionResult> GetNotifications()
        {
            var userId = User.FindFirst("id")?.Value;
            var notifications = await _context.Notifications
                .Where(n => n.UserId == userId)
                .OrderByDescending(n => n.CreatedAt)
                .ToListAsync();

            return Ok(notifications);
        }

        /// <summary>
        /// Пометить уведомление как прочитанное.

```

```

/// </summary>
/// <param name="id">ID уведомления.</param>
/// <returns>Результат операции.</returns>
[HttpPost("read/{id}")]
[Authorize]
public async Task<IActionResult> MarkAsRead(int id)
{
    var userId = User.FindFirst("id")?.Value;
    var notification = await _context.Notifications
        .FirstOrDefaultAsync(n => n.Id == id && n.UserId == userId);

    if (notification == null)
    {
        return NotFound("Уведомление не найдено.");
    }

    notification.IsRead = true;
    await _context.SaveChangesAsync();

    return Ok("Уведомление помечено как прочитанное.");
}

/// <summary>
/// Удалить уведомление.
/// </summary>
[HttpDelete("{id}")]
[Authorize]
public async Task<IActionResult> DeleteNotification(int id)
{
    var userId = User.FindFirst("id")?.Value;
    var notification = await _context.Notifications
        .FirstOrDefaultAsync(n => n.Id == id && n.UserId == userId);

    if (notification == null)
    {
        return NotFound("Уведомление не найдено.");
    }

    _context.Notifications.Remove(notification);
    await _context.SaveChangesAsync();

    return Ok("Уведомление удалено.");
}
}
}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Services;

namespace CryptoExchangeTrainingAPI.Controllers
{
    /// <summary>

```

```

/// Контроллер для получения рыночных данных.
/// </summary>
[ApiController]
[Route("api/[controller]")]
public class MarketDataController : ControllerBase
{
    private readonly IMarketService _marketService;

    /// <summary>
    /// Конструктор MarketDataController.
    /// </summary>
    /// <param name="marketService">Сервис для получения рыночных данных.</param>
    public MarketDataController(IMarketService marketService)
    {
        _marketService = marketService;
    }

    /// <summary>
    /// Получить текущую цену для торговой пары.
    /// </summary>
    /// <param name="pair">Торговая пара (например, BTCUSDT).</param>
    /// <returns>Цена для указанной пары.</returns>
    /// <response code="200">Цена успешно получена.</response>
    /// <response code="400">Ошибка при получении рыночных данных.</response>
    [HttpGet("{pair}")]
    public async Task<IActionResult> GetMarketPrice(string pair)
    {
        try
        {
            var price = await _marketService.GetPriceAsync(pair.ToUpper());
            return Ok(new { Pair = pair.ToUpper(), Price = price });
        }
        catch (Exception ex)
        {
            return BadRequest($"Ошибка при получении рыночных данных: {ex.Message}");
        }
    }
}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Services;
using System;
using System.Threading.Tasks;

namespace CryptoExchangeTrainingAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ChartController : ControllerBase
    {
        private readonly IMarketService _marketService;
    }
}

```

```

public ChartController(IMarketService marketService)
{
    _marketService = marketService;
}

[HttpGet("historical/{pair}")]
public async Task<IActionResult> GetHistoricalData(string pair, DateTime? startTime, DateTime?
endTime)
{
    if (startTime == null)
    {
        startTime = DateTime.UtcNow.AddDays(-7);
    }
    if (endTime == null)
    {
        endTime = DateTime.UtcNow;
    }

    try
    {
        var historicalData = await _marketService.GetHistoricalDataAsync(pair.ToUpper(),
startTime.Value, endTime.Value);
        return Ok(historicalData);
    }
    catch (Exception ex)
    {
        return BadRequest($"Ошибка при получении исторических данных: {ex.Message}");
    }
}
}

using Microsoft.AspNetCore.Mvc;
using CryptoExchangeTrainingAPI.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace CryptoExchangeTrainingAPI.Controllers
{
    /// <summary>
    /// Контроллер для управления аутентификацией и регистрацией пользователей.
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;

```

```

private readonly IConfiguration _configuration;

/// <summary>
/// Конструктор AuthController.
/// </summary>
/// <param name="userManager">Менеджер пользователей Identity.</param>
/// <param name="signInManager">Менеджер для управления входом пользователей.</param>
/// <param name="configuration">Конфигурация приложения.</param>
public AuthController(UserManager<User> userManager, SignInManager<User> signInManager,
IConfiguration configuration)
{
    _userManager = userManager;
    _signInManager = signInManager;
    _configuration = configuration;
}

/// <summary>
/// Регистрация нового пользователя.
/// </summary>
/// <param name="model">Данные для регистрации пользователя.</param>
/// <returns>Результат регистрации.</returns>
/// <response code="200">Пользователь успешно зарегистрирован.</response>
/// <response code="400">Некорректные данные для регистрации.</response>
[HttpPost("register")]
public async Task<IActionResult> Register([FromBody] RegisterDto model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(new
        {
            Success = false,
            Message = "Неверные данные."
        });
    }

    var userExists = await _userManager.FindByEmailAsync(model.Email);
    if (userExists != null)
    {
        return BadRequest(new
        {
            Success = false,
            Errors = new List<string> { "Email already registered" }
        });
    }

    var user = new User
    {
        UserName = model.Email,
        Email = model.Email,
        Balance = 10000.00m // Начальный баланс
    };
}

```

```

var result = await _userManager.CreateAsync(user, model.Password);

if (!result.Succeeded)
{
    var errors = result.Errors.Select(e => e.Description).ToList();
    return BadRequest(new
    {
        Success = false,
        Errors = errors
    });
}

return Ok(new
{
    Success = true,
    Message = "Пользователь успешно зарегистрирован."
});
}

/// <summary>
/// Авторизация пользователя.
/// </summary>
/// <param name="model">Данные для входа пользователя.</param>
/// <returns>JWT токен и информация о пользователе.</returns>
/// <response code="200">Успешная авторизация.</response>
/// <response code="400">Некорректные данные для входа.</response>
/// <response code="401">Неверный email или пароль.</response>
[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] LoginDto model)
{
    if (!ModelState.IsValid) return BadRequest("Неверные данные.");

    var user = await _userManager.FindByEmailAsync(model.Email);

    if (user == null)
    {
        return Unauthorized("Неверный email или пароль.");
    }

    var result = await _signInManager.PasswordSignInAsync(user, model.Password, false, false);

    if (!result.Succeeded)
    {
        return Unauthorized("Неверный email или пароль.");
    }

    // Генерация JWT-токена
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Id),

```

```

        new Claim(JwtRegisteredClaimNames.Email, user.Email),
        new Claim("id", user.Id)
    };

    // Проверка на null для конфигурационных значений
    var jwtSettings = _configuration.GetSection("JwtSettings");
    var secretKey = jwtSettings["Secret"];
    if (string.IsNullOrEmpty(secretKey))
    {
        return StatusCode(500, "JWT Secret Key не настроен.");
    }

    var issuer = jwtSettings["Issuer"];
    if (string.IsNullOrEmpty(issuer))
    {
        return StatusCode(500, "JWT Issuer не настроен.");
    }

    var audience = jwtSettings["Audience"];
    if (string.IsNullOrEmpty(audience))
    {
        return StatusCode(500, "JWT Audience не настроен.");
    }

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    var token = new JwtSecurityToken(
        issuer: issuer,
        audience: audience,
        claims: claims,
        expires: DateTime.Now.AddMinutes(int.Parse(jwtSettings["ExpiryMinutes"])),
        signingCredentials: creds
    );

    return Ok(new
    {
        Token = new JwtSecurityTokenHandler().WriteToken(token),
        Email = user.Email,
        Balance = user.Balance
    });
}
}
}
using CryptoExchangeTrainingAPI.Data;
using CryptoExchangeTrainingAPI.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace CryptoExchangeTrainingAPI.Controllers
{
    [ApiController]

```



```

[Route("api/[controller]")]
public class AssetController : ControllerBase
{
    private readonly ApplicationDbContext _context;

    public AssetController(ApplicationDbContext context)
    {
        _context = context;
    }

    // Получить список активов пользователя
    [HttpGet]
    public async Task<IActionResult> GetUserAssets()
    {
        var userId = User.Identity?.Name ?? "guest"; // Поддержка для авторизованных и гостевых
        пользователей

        var userAssets = await _context.UserAssets
            .Include(a => a.Token)
            .Where(a => a.UserId == userId)
            .ToListAsync();

        if (!userAssets.Any())
        {
            return Ok(new List<object>());
        }

        return Ok(userAssets.Select(a => new
        {
            Asset = a.Token.Name,
            Balance = a.Balance
        }));
    }

    // Пополнение токена
    [HttpPost("deposit")]
    public async Task<IActionResult> Deposit([FromBody] DepositRequestDto request)
    {
        if (request.Amount < 10 || request.Amount > 1000)
        {
            return BadRequest(new { Message = "Сумма пополнения должна быть от 10 до 1000." });
        }

        var userId = User.Identity?.Name ?? "guest";

        var userAsset = await _context.UserAssets
            .Include(a => a.Token)
            .FirstOrDefaultAsync(a => a.UserId == userId && a.Token.Symbol == request.Asset);

        if (userAsset == null)
        {

```

```

        return NotFound(new { Message = "Актив не найден." });
    }

    userAsset.Balance += request.Amount;
    userAsset.UpdatedAt = DateTime.UtcNow;

    await _context.SaveChangesAsync();

    return Ok(new { Message = $"Баланс токена {request.Asset} пополнен на {request.Amount}." });
}
}

using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;

namespace CryptoExchangeTrainingAPI.Services
{
    public class JwtTokenService
    {
        private readonly IConfiguration _configuration;

        public JwtTokenService(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public string GenerateToken(string userId, string email)
        {
            var jwtSettings = _configuration.GetSection("JwtSettings");

            var secretKey = jwtSettings["Secret"];
            var issuer = jwtSettings["Issuer"];
            var audience = jwtSettings["Audience"];
            var expiryMinutes = int.Parse(jwtSettings["ExpiryMinutes"]);

            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub, userId),
                new Claim(JwtRegisteredClaimNames.Email, email),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            };

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

```

```

        var token = new JwtSecurityToken(
            issuer: issuer,
            audience: audience,
            claims: claims,
            expires: DateTime.UtcNow.AddMinutes(expiryMinutes),
            signingCredentials: creds
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
}
using System;
using Microsoft.Extensions.Caching.Memory;

namespace CryptoExchangeTrainingAPI.Services
{
    public interface ICacheService
    {
        T? Get<T>(string key);
        void Set<T>(string key, T value, TimeSpan duration);
    }

    public class CacheService : ICacheService
    {
        private readonly MemoryCache _cache = new MemoryCache(new MemoryCacheOptions());

        public T? Get<T>(string key)
        {
            return _cache.TryGetValue(key, out T value) ? value : default;
        }

        public void Set<T>(string key, T value, TimeSpan duration)
        {
            _cache.Set(key, value, duration);
        }
    }
}
using System.Net.Http;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace CryptoExchangeTrainingAPI.Services
{
    public interface IHttpRequestService
    {
        Task<T> GetAsync<T>(string url) where T : class;
    }

    public class HttpRequestService : IHttpRequestService
    {

```

```

private readonly HttpClient _httpClient;

public HttpRequestService(HttpClient httpClient)
{
    _httpClient = httpClient;
}

public async Task<T> GetAsync<T>(string url) where T : class
{
    var response = await _httpClient.GetAsync(url);
    if (!response.IsSuccessStatusCode)
    {
        throw new Exception($"HTTP запрос не успешен. Код: {response.StatusCode}, URL: {url}");
    }

    var content = await response.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<T>(content) ?? throw new Exception("Ошибка
десериализации.");
}
}
}
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Extensions.Caching.Memory;
using Newtonsoft.Json;
using System.Collections.Generic;
using System;
using System.Linq;
using CryptoExchangeTrainingAPI.Models;
using CryptoExchangeTrainingAPI.Data;
using Microsoft.EntityFrameworkCore;
using System.Globalization;

namespace CryptoExchangeTrainingAPI.Services
{
    public interface IMarketService
    {
        Task<decimal> GetPriceAsync(string pair);
        Task<List<HistoricalData>> GetHistoricalDataAsync(string pair, DateTime startTime, DateTime
endTime);
        Task<OrderBookResponse> GetOrderBookAsync(string pair);
    }

    public class MarketService : IMarketService
    {
        private readonly MemoryCache _cache = new MemoryCache(new MemoryCacheOptions());
        private readonly HttpClient _httpClient;
        private readonly ApplicationDbContext _context;
        public MarketService(HttpClient httpClient, ApplicationDbContext context)
        {
            _httpClient = httpClient;

```

```

        _context = context;
    }

    public async Task<decimal> GetPriceAsync(string pair)
    {
        if (!_cache.TryGetValue(pair, out decimal price))
        {
            // Если в кэше нет данных, делаем запрос
            var response = await
_httpClient.GetAsync($"https://api.binance.com/api/v3/ticker/price?symbol={pair}");
            if (!response.IsSuccessStatusCode)
            {
                throw new Exception("Не удалось получить данные о рынке.");
            }
            var content = await response.Content.ReadAsStringAsync();
            var marketData = JsonConvert.DeserializeObject<MarketDataResponse>(content);
            price = marketData.Price;

            _cache.Set(pair, price, TimeSpan.FromSeconds(10));
        }

        return price;
    }

    public async Task<OrderBookResponse> GetOrderBookAsync(string pair)
    {
        var response = await
_httpClient.GetAsync($"https://api.binance.com/api/v3/depth?symbol={pair}&limit=10");
        if (!response.IsSuccessStatusCode)
        {
            throw new Exception("Не удалось получить данные из OrderBook.");
        }
        var content = await response.Content.ReadAsStringAsync();
        var orderBook = JsonConvert.DeserializeObject<OrderBookResponse>(content);
        return orderBook;
    }

    public async Task<List<HistoricalData>> GetHistoricalDataAsync(string pair, DateTime startTime,
DateTime endTime)
    {
        var data = new List<HistoricalData>();
        var interval = "1h";
        long startTimeUnix = ((DateTimeOffset)startTime).ToUnixTimeMilliseconds();
        long endTimeUnix = ((DateTimeOffset)endTime).ToUnixTimeMilliseconds();

        var response = await
_httpClient.GetAsync($"https://api.binance.com/api/v3/klines?symbol={pair}&interval={interval}&startTime={startTimeUnix}&endTime={endTimeUnix}");
        if (!response.IsSuccessStatusCode)
        {
            throw new Exception("Не удалось получить исторические данные с рынка.");
        }
    }

```

```

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

var klines = JsonConvert.DeserializeObject<List<List<object>>>>(content);
if (klines != null)
{
    data = klines.Select(k => new HistoricalData
    {
        Pair = pair,
        Timestamp = DateTimeOffset.FromUnixTimeMilliseconds((long)k[0]).DateTime,
        Price = decimal.Parse((string)k[4], CultureInfo.InvariantCulture),
    }).ToList();
}
return data;
}

private class MarketDataResponse
{
    [JsonProperty("symbol")]
    public string Symbol { get; set; } = null!;

    [JsonProperty("price")]
    public decimal Price { get; set; }
}

}

public class OrderBookResponse
{
    [JsonProperty("lastUpdateId")]
    public long LastUpdateId { get; set; }

    [JsonProperty("bids")]
    public List<List<string>> Bids { get; set; } = null!;

    [JsonProperty("asks")]
    public List<List<string>> Asks { get; set; } = null!;
}

}
using CryptoExchangeTrainingAPI.Data;
using CryptoExchangeTrainingAPI.Models;

namespace CryptoExchangeTrainingAPI.Services
{
    public interface INotificationService
    {
        Task CreateNotificationAsync(string userId, string message);
    }

    public class NotificationService : INotificationService
    {
        private readonly ApplicationDbContext _context;
    }
}

```

```

public NotificationService(ApplicationDbContext context)
{
    _context = context;
}

public async Task CreateNotificationAsync(string userId, string message)
{
    var notification = new Notification
    {
        UserId = userId,
        Message = message,
        IsRead = false,
        CreatedAt = DateTime.UtcNow
    };

    await _context.Notifications.AddAsync(notification);
    await _context.SaveChangesAsync();
}

public async Task CreateNotificationsAsync(List<string> userIds, string message)
{
    var notifications = userIds.Select(userId => new Notification
    {
        UserId = userId,
        Message = message,
        IsRead = false,
        CreatedAt = DateTime.UtcNow
    }).ToList();

    await _context.Notifications.AddRangeAsync(notifications);
    await _context.SaveChangesAsync();
}
}

using Microsoft.AspNetCore.SignalR;

namespace CryptoExchangeTrainingAPI.Services
{
    public class PriceHub : Hub
    {
        {
            public async Task UpdatePrices(Dictionary<string, decimal> prices)
            {
                await Clients.All.SendAsync("ReceivePriceUpdates", prices);
            }
        }
    }
}

namespace CryptoExchangeTrainingUI.Configuration
{
    public class ApiSettings
    {

```

```

        public string BaseUrl { get; set; } = string.Empty;
        public int TimeoutSeconds { get; set; } = 30;
    }
}

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Configuration;
using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components.Authorization;
using CryptoExchangeTrainingUI.Configuration;
using CryptoExchangeTrainingUI.Services.Authentication;
using CryptoExchangeTrainingUI.Services.Common;

namespace CryptoExchangeTrainingUI.Extensions
{
    public static class ServiceCollectionExtensions
    {
        public static IServiceCollection AddApplicationServices(
            this IServiceCollection services,
            IConfiguration configuration)
        {
            // Конфигурация
            services.Configure<ApiSettings>(
                configuration.GetSection("ApiSettings"));

            // Сервисы Blazored
            services.AddBlazoredLocalStorage();

            // Аутентификация
            services.AddAuthorizationCore();
            services.AddScoped<AuthenticationStateProvider, CustomAuthenticationStateProvider>();

            // Сервисы приложения
            services.AddScoped<IAuthService, AuthService>();
            services.AddScoped<IApiErrorHandler, ApiErrorHandler>();

            return services;
        }
    }
}

namespace CryptoExchangeTrainingUI.Helpers
{
    public static class Constants
    {
        public static class Storage
        {
            public const string AuthToken = "authToken";
            public const string RefreshToken = "refreshToken";
            public const string UserData = "userData";
        }
    }
}

```



```

public static class Api
{
    public const string Login = "api/auth/login";
    public const string Register = "api/auth/register";
    public const string RefreshToken = "api/auth/refresh";
}

public static class Routes
{
    public const string Login = "/login";
    public const string Register = "/register";
    public const string Home = "/";
    public const string Profile = "/profile";
}
}

using System.Security.Claims;
using System.Text.Json;

namespace CryptoExchangeTrainingUI.Helpers
{
    public static class JwtParser
    {
        public static IEnumerable<Claim> ParseClaimsFromJwt(string jwt)
        {
            var claims = new List<Claim>();
            var payload = jwt.Split('.')[1];
            var jsonBytes = ParseBase64WithoutPadding(payload);
            var keyValuePairs = JsonSerializer.Deserialize<Dictionary<string, object>>(jsonBytes);

            if (keyValuePairs != null)
            {
                claims.AddRange(keyValuePairs.Select(kvp => new Claim(kvp.Key, kvp.Value.ToString() ??
string.Empty)));
            }

            return claims;
        }

        private static byte[] ParseBase64WithoutPadding(string base64)
        {
            switch (base64.Length % 4)
            {
                case 2: base64 += "=="; break;
                case 3: base64 += "="; break;
            }
            return Convert.FromBase64String(base64);
        }
    }
}

```

```

namespace CryptoExchangeTrainingUI.Helpers
{
    public static class ValidationHelper
    {
        public static bool IsValidEmail(string email)
        {
            try
            {
                var addr = new System.Net.Mail.MailAddress(email);
                return addr.Address == email;
            }
            catch
            {
                return false;
            }
        }

        public static bool IsValidPassword(string password)
        {
            if (string.IsNullOrEmpty(password)) return false;

            var hasNumber = password.Any(char.IsDigit);
            var hasUpperChar = password.Any(char.IsUpper);
            var hasLowerChar = password.Any(char.IsLower);
            var hasMinimumChars = password.Length >= 6;

            return hasNumber && hasUpperChar && hasLowerChar && hasMinimumChars;
        }
    }
}
using System.Text.Json.Serialization;
using CryptoExchangeTrainingUI.Models.User;

namespace CryptoExchangeTrainingUI.Models.Authentication
{
    public class AuthResponse
    {
        [JsonPropertyName("token")]
        public string Token { get; init; } = string.Empty;

        [JsonPropertyName("refreshToken")]
        public string RefreshToken { get; init; } = string.Empty;

        [JsonPropertyName("success")]
        public bool Success { get; init; }

        [JsonPropertyName("message")]
        public string Message { get; init; } = string.Empty;

        [JsonPropertyName("email")]
        public string Email { get; init; } = string.Empty;
    }
}

```

```

        [JsonPropertyName("errors")]
        public List<string> Errors { get; init; } = new();
    }

}
using System.Text.Json.Serialization;

namespace CryptoExchangeTrainingUI.Models.Authentication
{
    public class ErrorDetails
    {
        [JsonPropertyName("message")]
        public string Message { get; init; } = string.Empty;

        [JsonPropertyName("statusCode")]
        public int StatusCode { get; init; }
    }
}
using System.ComponentModel.DataAnnotations;

namespace CryptoExchangeTrainingUI.Models.Authentication
{
    public class LoginRequest
    {
        private string _email = string.Empty;
        private string _password = string.Empty;

        [Required(ErrorMessage = "Email is required")]
        [EmailAddress(ErrorMessage = "Invalid email format")]
        [MaxLength(100, ErrorMessage = "Email cannot exceed 100 characters")]
        public string Email
        {
            get => _email;
            set => _email = value?.Trim() ?? string.Empty;
        }

        [Required(ErrorMessage = "Password is required")]
        [MinLength(6, ErrorMessage = "Password must be at least 6 characters")]
        [MaxLength(100, ErrorMessage = "Password cannot exceed 100 characters")]
        public string Password
        {
            get => _password;
            set => _password = value ?? string.Empty;
        }
    }
}
using System.ComponentModel.DataAnnotations;

namespace CryptoExchangeTrainingUI.Models.Authentication
{

```

```

public class RegisterRequest
{
    private string _email = string.Empty;
    private string _password = string.Empty;
    private string _confirmPassword = string.Empty;

    [Required(ErrorMessage = "Email is required")]
    [EmailAddress(ErrorMessage = "Invalid email format")]
    [MaxLength(100, ErrorMessage = "Email cannot exceed 100 characters")]
    public string Email
    {
        get => _email;
        set => _email = value?.Trim() ?? string.Empty;
    }

    [Required(ErrorMessage = "Password is required")]
    [MinLength(6, ErrorMessage = "Password must be at least 6 characters")]
    [MaxLength(100, ErrorMessage = "Password cannot exceed 100 characters")]
    [RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$",
        ErrorMessage = "Password must contain at least one uppercase letter, one lowercase letter, one
number and one special character")]
    public string Password
    {
        get => _password;
        set => _password = value ?? string.Empty;
    }

    [Required(ErrorMessage = "Confirm Password is required")]
    [Compare(nameof(Password), ErrorMessage = "Passwords do not match")]
    public string ConfirmPassword
    {
        get => _confirmPassword;
        set => _confirmPassword = value ?? string.Empty;
    }
}

}

using System.Text.Json.Serialization;

namespace CryptoExchangeTrainingUI.Models.Authentication
{
    public class LoginResponse
    {
        [JsonPropertyName("token")]
        public string Token { get; init; } = string.Empty;
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class DepositRequestDto

```

```

    {
        public string Asset { get; set; } = string.Empty;
        public decimal Amount { get; set; }
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class DepositResponseDto
    {
        public decimal Balance { get; set; }

    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class AssetInfo
    {
        public string Symbol { get; set; }
        public decimal Quantity { get; set; }
        public decimal CurrentPrice { get; set; }
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class TokenDto
    {
        public string Symbol { get; set; } = string.Empty; // СИМВОЛ токена (BTC, ETH)
        public string Name { get; set; } = string.Empty; // Название токена (Bitcoin, Ethereum)
        public decimal Price { get; set; } // Цена токена
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class TradeDto
    {
        public int Id { get; set; }
        public string Pair { get; set; } = string.Empty;
        public string Type { get; set; } = string.Empty; // "buy" или "sell"
        public int Leverage { get; set; }
        public decimal Amount { get; set; }
        public decimal EntryPrice { get; set; }
        public decimal? StopLoss { get; set; }
        public decimal? TakeProfit { get; set; }
        public string Status { get; set; } = string.Empty; // "open" или "closed"
        public DateTime OpenedAt { get; set; }
        public DateTime? ClosedAt { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;

```

```

namespace CryptoExchangeTrainingUI.Models.User
{
    public class TradeRequest
    {
        [Required]
        public string Pair { get; set; }

        [Required]
        public string Type { get; set; }

        [Required]
        [Range(0, double.MaxValue)]
        public decimal Amount { get; set; }

        [Required]
        [Range(1, 100)]
        public int Leverage { get; set; }

        public decimal? StopLoss { get; set; }
        public decimal? TakeProfit { get; set; }
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class UserAssetDto
    {
        public string Asset { get; set; } = string.Empty; // Название криптовалюты (BTC, ETH)
        public decimal Balance { get; set; } // Баланс пользователя
    }
}

namespace CryptoExchangeTrainingUI.Models.User
{
    public class UserDto
    {
        public string Email { get; set; }
        public decimal Balance { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime? LastLoginAt { get; set; }
        public string Token { get; set; }
    }
}

using Blazored.LocalStorage;
using CryptoExchangeTrainingUI.Helpers;
using CryptoExchangeTrainingUI.Models.Authentication;
using Microsoft.AspNetCore.Components.Authorization;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Security.Claims;
using CryptoExchangeTrainingUI.Services.Common;

```

```

using System.Text.Json;
using System.Text.Json.Serialization;
using CryptoExchangeTrainingUI.Models.User;
namespace CryptoExchangeTrainingUI.Services.Authentication
{
    public class AuthService : IAuthService
    {
        private readonly HttpClient _httpClient;
        private readonly ILocalStorageService _localStorage;
        private readonly AuthenticationStateProvider _authStateProvider;
        private readonly ILogger<AuthService> _logger;
        private readonly IApiErrorHandler _errorHandler;

        public AuthService(
            HttpClient httpClient,
            ILocalStorageService localStorage,
            AuthenticationStateProvider authStateProvider,
            ILogger<AuthService> logger,
            IApiErrorHandler errorHandler)
        {
            _httpClient = httpClient ?? throw new ArgumentNullException(nameof(httpClient));
            _localStorage = localStorage ?? throw new ArgumentNullException(nameof(localStorage));
            _authStateProvider = authStateProvider ?? throw new
ArgumentNullException(nameof(authStateProvider));
            _logger = logger ?? throw new ArgumentNullException(nameof(logger));
            _errorHandler = errorHandler ?? throw new ArgumentNullException(nameof(errorHandler));
        }

        public async Task<string?> GetToken()
        {
            try
            {
                return await _localStorage.GetItemAsync<string>(Constants.Storage.AuthToken);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error retrieving token");
                return null;
            }
        }

        public async Task<bool> IsAuthenticated()
        {
            try
            {
                var token = await _localStorage.GetItemAsync<string>(Constants.Storage.AuthToken);
                if (string.IsNullOrEmpty(token))
                    return false;

                var handler = new JwtSecurityTokenHandler();
                var jsonToken = handler.ReadToken(token) as JwtSecurityToken;
            }
        }
    }
}

```

```

        if (jsonToken == null)
            return false;

        return jsonToken.ValidTo > DateTime.UtcNow;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error checking authentication status");
        return false;
    }
}

public async Task<AuthResponse> Login(LoginRequest loginRequest)
{
    try
    {
        var response = await _httpClient.PostAsJsonAsync("api/auth/login", loginRequest);

        if (response.IsSuccessStatusCode)
        {
            var userDto = await response.Content.ReadFromJsonAsync<UserDto>();
            if (userDto != null)
            {
                await _localStorage.SetItemAsync("authToken", userDto.Token);
                _httpClient.DefaultRequestHeaders.Authorization =
                    new AuthenticationHeaderValue("Bearer", userDto.Token);
                ((CustomAuthenticationStateProvider)_authStateProvider)
                    .NotifyUserAuthentication(userDto.Token);

                return new AuthResponse { Success = true };
            }
        }

        return new AuthResponse
        {
            Success = false,
            Message = "Invalid email or password"
        };
    }
    catch
    {
        return new AuthResponse { Success = false };
    }
}

public async Task<AuthResponse> Register(RegisterRequest registerRequest)
{
    try
    {
        var response = await _httpClient.PostAsJsonAsync("api/auth/register", registerRequest);
    }
}

```



```

        if (response.IsSuccessStatusCode)
        {
            var authResponse = await response.Content.ReadFromJsonAsync<AuthResponse>();
            if (authResponse != null && authResponse.Success)
            {
                return new AuthResponse
                {
                    Success = true,
                    Message = "Registration successful"
                };
            }
            // If success is false, but we got an AuthResponse, use its details
            return authResponse ?? new AuthResponse { Success = false, Message = "An unexpected error
occurred" };
        }
        else
        {
            var errors = await _errorHandler.HandleApiError(response);
            return new AuthResponse
            {
                Success = false,
                Message = "Registration failed",
                Errors = errors
            };
        }
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error during registration");
        return new AuthResponse
        {
            Success = false,
            Message = "An error occurred while attempting to register",
            Errors = new List<string> { ex.Message }
        };
    }
}

public async Task Logout()
{
    try
    {
        await _localStorage.RemoveItemAsync(Constants.Storage.AuthToken);
        await _localStorage.RemoveItemAsync(Constants.Storage.RefreshToken);
        _httpClient.DefaultRequestHeaders.Authorization = null;
        ((CustomAuthenticationStateProvider)_authStateProvider).NotifyUserLogout();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error during logout");
    }
}

```

```

        throw;
    }
}

public async Task<string> GetUserDisplayName()
{
    var token = await GetToken();
    if (string.IsNullOrEmpty(token))
        return string.Empty;

    try
    {
        var handler = new JwtSecurityTokenHandler();
        var jsonToken = handler.ReadToken(token) as JwtSecurityToken;
        var email = jsonToken?.Claims.FirstOrDefault(c => c.Type == "email")?.Value ?? string.Empty;
        return email.Split('@')[0];
    }
    catch
    {
        return string.Empty;
    }
}

public async Task<AuthResponse> RefreshToken()
{
    try
    {
        var refreshToken = await _localStorage.GetItemAsync<string>(Constants.Storage.RefreshToken);
        if (string.IsNullOrEmpty(refreshToken))
        {
            _logger.LogWarning("Refresh token not found in storage");
            return new AuthResponse
            {
                Success = false,
                Message = "No refresh token available"
            };
        }

        var response = await _httpClient.PostAsJsonAsync(Constants.Api.RefreshToken,
            new { RefreshToken = refreshToken });

        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadFromJsonAsync<AuthResponse>();
            if (result?.Success == true && !string.IsNullOrEmpty(result.Token))
            {
                await _localStorage.SetItemAsync(Constants.Storage.AuthToken, result.Token);
                await _localStorage.SetItemAsync(Constants.Storage.RefreshToken, result.RefreshToken);

                _httpClient.DefaultRequestHeaders.Authorization =

```

```

        new AuthenticationHeaderValue("Bearer", result.Token);

        return result;
    }
}

var errors = await _errorHandler.HandleApiError(response);
return new AuthResponse
{
    Success = false,
    Message = "Token refresh failed",
    Errors = errors
};
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error during token refresh");
    return new AuthResponse
    {
        Success = false,
        Message = "An unexpected error occurred during token refresh",
        Errors = new List<string> { ex.Message }
    };
}
}
}

using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components.Authorization;
using System.Net.Http.Headers;
using System.Security.Claims;
using System.Text.Json;

public class CustomAuthenticationStateProvider : AuthenticationStateProvider
{
    private readonly ILocalStorageService _localStorage;
    private readonly HttpClient _httpClient;

    public CustomAuthenticationStateProvider(ILocalStorageService localStorage, HttpClient httpClient)
    {
        _localStorage = localStorage;
        _httpClient = httpClient;
    }

    public override async Task<AuthenticationState> GetAuthenticationStateAsync()
    {
        var token = await _localStorage.GetItemAsync<string>("authToken");

        if (string.IsNullOrEmpty(token))
        {
            return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));
        }
    }
}

```

```

    }

    _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

    return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(ParseClaimsFromJwt(token),
"jwt")));
}

private IEnumerable<Claim> ParseClaimsFromJwt(string jwt)
{
    var claims = new List<Claim>();
    var payload = jwt.Split('.')[1];
    var jsonBytes = ParseBase64WithoutPadding(payload);
    var keyValuePairs = JsonSerializer.Deserialize<Dictionary<string, object>>(jsonBytes);

    if (keyValuePairs != null)
    {
        keyValuePairs.TryGetValue(ClaimTypes.Role, out object roles);

        if (roles != null)
        {
            {
                if (roles.ToString().Trim().StartsWith("["))
                {
                    var parsedRoles = JsonSerializer.Deserialize<string[]>(roles.ToString());
                    foreach (var parsedRole in parsedRoles)
                    {
                        claims.Add(new Claim(ClaimTypes.Role, parsedRole));
                    }
                }
                else
                {
                    claims.Add(new Claim(ClaimTypes.Role, roles.ToString()));
                }
            }

            keyValuePairs.Remove(ClaimTypes.Role);
        }

        claims.AddRange(keyValuePairs.Select(kvp => new Claim(kvp.Key, kvp.Value.ToString())));
    }

    return claims;
}

private byte[] ParseBase64WithoutPadding(string base64)
{
    switch (base64.Length % 4)
    {
        case 2: base64 += "=="; break;
        case 3: base64 += "="; break;
    }
    return Convert.FromBase64String(base64);
}

```

```

    }

    public void NotifyUserAuthentication(string token)
    {
        var authenticatedUser = new ClaimsPrincipal(new ClaimsIdentity(ParseClaimsFromJwt(token), "jwt"));
        var authState = Task.FromResult(new AuthenticationState(authenticatedUser));
        NotifyAuthenticationStateChanged(authState);
    }

    public void NotifyUserLogout()
    {
        var anonymousUser = new ClaimsPrincipal(new ClaimsIdentity());
        var authState = Task.FromResult(new AuthenticationState(anonymousUser));
        NotifyAuthenticationStateChanged(authState);
    }
}
using CryptoExchangeTrainingUI.Models.Authentication;

namespace CryptoExchangeTrainingUI.Services.Authentication
{
    public interface IAuthService
    {
        Task<AuthResponse> Login(LoginRequest loginRequest);
        Task<AuthResponse> Register(RegisterRequest registerRequest);
        Task<bool> IsAuthenticated();
        Task Logout();
        Task<string?> GetToken();
        Task<string> GetUserDisplayName();
    }
}

using CryptoExchangeTrainingUI.Models.User;

namespace CryptoExchangeTrainingUI.Services.Trade
{
    public interface ITradeService
    {
        Task<List<TradeDto>> GetActiveTradesAsync();
        Task<TradeDto> OpenTradeAsync(TradeRequest request);
        Task<TradeDto> CloseTradeAsync(int tradeId);
        Task<List<TradeDto>> GetTradeHistoryAsync();
    }
}

using Blazored.LocalStorage;
using CryptoExchangeTrainingUI.Models.User;
using Microsoft.AspNetCore.Components.Authorization;
using System.Net.Http.Json;
using System.Linq; // Добавьте это для использования методов расширения LINQ

```

```

namespace CryptoExchangeTrainingUI.Services.Trade
{
    public class TradeService : ITradeService
    {
        private readonly HttpClient _httpClient;
        private readonly ILocalStorageService _localStorage;
        private readonly AuthenticationStateProvider _authStateProvider;

        public TradeService(
            HttpClient httpClient,
            ILocalStorageService localStorage,
            AuthenticationStateProvider authStateProvider)
        {
            _httpClient = httpClient;
            _localStorage = localStorage;
            _authStateProvider = authStateProvider;
        }

        public async Task<List<TradeDto>> GetActiveTradesAsync()
        {
            try
            {
                var response = await _httpClient.GetFromJsonAsync<List<TradeDto>>("api/Trade/history");
                return response?.Where(t => t.Status == "open").ToList() ?? new List<TradeDto>();
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при получении активных сделок", ex);
            }
        }

        public async Task<TradeDto> OpenTradeAsync(TradeRequest request)
        {
            try
            {
                var response = await _httpClient.PostAsJsonAsync("api/Trade/open", request);
                if (response.IsSuccessStatusCode)
                {
                    var result = await response.Content.ReadFromJsonAsync<TradeDto>();
                    // Обновляем токен с новым балансом
                    var token = response.Headers.GetValues("new-token").FirstOrDefault();
                    if (!string.IsNullOrEmpty(token))
                    {
                        await _localStorage.SetItemAsync("authToken", token);
                        ((CustomAuthenticationStateProvider)_authStateProvider).NotifyUserAuthentication(token);
                    }
                    return result;
                }
                throw new Exception(await response.Content.ReadAsStringAsync());
            }
        }
    }
}

```

```

        catch (Exception ex)
        {
            throw new Exception("Ошибка при открытии сделки", ex);
        }
    }

    public async Task<TradeDto> CloseTradeAsync(int tradeId)
    {
        try
        {
            var response = await _httpClient.PostAsJsonAsync($"api/Trade/close/{tradeId}", new { tradeId });
            if (response.IsSuccessStatusCode)
            {
                var result = await response.Content.ReadFromJsonAsync<TradeDto>();

                if (response.Headers.Contains("new-token"))
                {
                    var token = response.Headers.GetValues("new-token").FirstOrDefault();
                    if (!string.IsNullOrEmpty(token))
                    {
                        await _localStorage.SetItemAsync("authToken", token);
                    }
                }

                ((CustomAuthenticationStateProvider)_authStateProvider).NotifyUserAuthentication(token);
            }
        }

        // Принудительно обновляем список активных сделок
        await GetActiveTradesAsync();

        return result;
    }
    var errorMessage = await response.Content.ReadAsStringAsync();
    throw new Exception($"Ошибка при закрытии сделки: {errorMessage}");
}
catch (Exception ex)
{
    throw new Exception($"Ошибка при закрытии сделки: {ex.Message}", ex);
}

}

public async Task<List<TradeDto>> GetTradeHistoryAsync()
{
    return await _httpClient.GetFromJsonAsync<List<TradeDto>>("api/Trade/history")
        ?? new List<TradeDto>();
}
}
}

```

ПРИЛОЖЕНИЕ Д
(обязательное)

Схема данных приложений