

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МО и ПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Система сбора и анализа данных крипторынка

Студент

(подпись, дата)

Агафонов Захар Геннадьевич
(ФИО полностью)

Руководитель

(подпись, дата)

Макарычев П. П.
(фамилия, инициалы)

Нормоконтролер

(подпись, дата)

Такташкин Д.В
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Кафедра

Вычислительной техники

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

1. Студент Агафонов Захар Геннадьевич гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Сервис сбора и анализа данных крипторынка

Тема утверждена приказом ПГУ № _____ от "___" _____ 2025 г.

3. Руководитель работы к.т.н., профессор кафедры «МОиПЭВМ» Макарычев П. П.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Требуется разработать приложение, обеспечивающее сбор и анализ данных крипторынка

2) Среда выполнения: операционная система macOS Sequoia 15.1

3) Языковые средства разработки: Golang, SQL, UML.

4) Инструментальные средства разработки: JetBrains Goland, Visual Studio Code, Docker, DataGrip

5) Технологии разработки:

б) Основные функции приложения:

а) регистрация и авторизация пользователей;

б) агрегация данных пользователем с криптобиржи

в) анализ средней амплитуды свечей

г) анализ соотношения бычьих и медвежьих свечей

д) анализ волатильности актива

е) анализ экстремальных значений цен

ж) анализ процента изменения цены

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) План разработки бюджета;

6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2025 – 16.04.2025	
2	Проектирование программных средств	20%	17.04.2025 – 26.04.2025	
3	Реализация программных средств	30%	27.04.2025 – 11.05.2025	
4	Тестирование программных средств	20%	12.05.2025 – 21.05.2025	
5	Оформление пояснительной записки	20%	22.05.2025 – 01.06.2025	

Дата выдачи задания «08» _____ апреля _____ 2025 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 87 листов, 38 рисунков, 13 таблиц, 21 использованный источник и 2 приложения.

GOLANG, JS, БАЗА ДАННЫХ, POSTGRES, KAFKA, CLICKHOUSE, MICROSOFT VISUAL STUDIO CODE, DOCKER.

Объект исследования – сервисы сбора и анализа крипторынка.

Цель выпускной квалификационной работы – разработать сервис сбора и анализа данных крипторынка.

Технологии разработки – разработка сервера приложений проводилась на языке программирования Golang, клиентская часть разрабатывалась на языке JavaScript с использованием фреймворка React.js, для создания сервера БД использовался PostgreSQL и Clickhouse. В качестве брокера сообщений используется Kafka. В качестве среды разработки использовался редактор программного кода VS Code и менеджер баз данных DataGrip.

Среда исполнения – Linux, Web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть системы сбора и анализа данных крипторынка. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08ВР211.17 ПЗ							
Изм.	Лист	№ докум.	Подп.	Дата								
Разраб.		Агафонов З.Г.			Система сбора и анализа данных крипторынка. Пояснительная записка				Лит.	Лист	Листов	
Пров.		Макарычев П. П.										
Реценз.												
Н. контр.		Такташкин Д.В										
Утверд.		Козлов А.Ю.										
					Группа 21ВИ1							

СОДЕРЖАНИЕ

Введение.....	6
1. Анализ сервисов сбора данных и анализа крипторынка.....	7
1.1 Анализ предметной области	7
1.2 Сравнительный анализ сервисов для аналитики крипторынка.....	8
1.2.1 Агрегатор рыночных данных CoinGeko	8
1.2.2 Платформа CoinMarketCap.....	9
1.2.3 Платформа CryptoCompare.....	11
1.2.4 Платформа Binance API.....	12
1.3 Сравнение сервисов сбора и анализа данных крипторынка.....	13
1.4 Метода анализа данных криптовалютного рынка	16
1.4.1 Средняя амплитуда свечей.....	17
1.4.2 Соотношение бычьих и медвежьих свечей	17
1.4.3 Анализ волатильности актива.....	18
1.4.4 Процент изменения цены	18
1.4.5 Экстремальные значения цены.....	19
1.5. Постановка задачи.....	21
1.6. Анализ требований.....	19
1.6.1 Функциональные требования	23
1.6.2 Нефункциональные требования	24
2. Проектирование сервиса сбора и анализа данных крипторынка.....	26
2.1 Архитектура сервиса.....	26
2.2 Проектирование структур данных	27
2.2.1 Логическая модель базы данных аутентификации.....	27
2.2.2 Физическая модель базы данных аутентификации	28
2.2.3 Проектирование таблиц под аналитические запросы	29
2.2.4 Проектирование топиков Kafka.....	32
2.3 Проектирование программных средств.....	32

2.3.1	Проектирования сервиса работы с API биржи.....	32
2.3.2	Проектирование сервиса аналитики	34
2.3.3	Проектирование сервиса аутентификации	36
2.4	Проектирование пользовательского интерфейса web-приложения	40
3.	Реализация программного обеспечения сервиса	45
4.	Контроль качества программного обеспечения.....	49
4.1	Функциональное тестирование.....	49
4.2	Модульное тестирование	52
4.3	Метрики кода.....	55
4.4	Планирование разработки и оценка бюджета	59
	Заключение	61
	Список использованных источников	62
	Приложение А	65
	Приложение Б	84

Введение

В настоящее время финансовая система претерпевает масштабные изменения под напором стремительного прогресса цифровых технологий. Одним из самых заметных и динамично развивающихся сегментов в этой сфере является рынок криптовалют. Такие цифровые активы, как Bitcoin, Ethereum и многие другие, прочно вошли в глобальное инвестиционное пространство и привлекают всё большее внимание со стороны частных инвесторов, корпораций и научного сообщества.

Крипторынок выделяется своей высокой волатильностью, круглосуточным режимом работы и огромным объемом поступающих в режиме реального времени данных. Для успешного анализа рыночной ситуации, формирования обоснованных инвестиционных стратегий и разработки алгоритмических торговых планов требуется надежная система, способная эффективно собирать, структурировать и интерпретировать информацию из разных источников, включая биржи, новостные агрегаторы, блокчейн-мониторы и другие.

В данной работе анализируется процесс создания программной системы, которая предназначена для автоматизации сбора, сохранения и анализа данных с криптовалютного сегмента. Особое внимание уделяется разработке архитектуры системы, выбору подходящих технологий и инструментов, а также методам обработки и представления полученной информации. Представленное решение может быть актуально для широкого круга пользователей, включая трейдеров, аналитиков, разработчиков торговых стратегий и специалистов в области блокчейн-экономики.

Целью выпускной квалификационной работы является разработка сервиса сбора и анализа данных крипторынка. Разработка включает в себя анализ требований, проектирование, реализация и тестирование программного обеспечения.

1. Анализ сервисов сбора данных и анализа крипторынка

1.1. Анализ предметной области

Данные криптовалютного рынка представляют собой совокупность информации, отражающей состояние цифровых активов и сопутствующих процессов. К таким данным относятся курсы криптовалют в реальном времени, объёмы торгов, данные о заявках, история сделок, данные блокчейна, а также метаданные об активах, токенах и биржах. Эти данные являются основой для проведения технического анализа, построения торговых стратегий, оценки рыночной динамики и принятия инвестиционных решений.

В отличие от классических финансовых рынков, где данные часто закрыты и доступны только через лицензированные платформы, крипторынок изначально формировался как открытая и децентрализованная среда. Большинство криптовалютных бирж, агрегаторов и блокчейн-обозревателей публикуют свои данные в открытом формате, предоставляя API-интерфейсы для получения актуальной информации в режиме реального времени. Это делает рынок доступным для автоматизированного анализа и использования сторонними разработчиками.

Хотя в индустрии нет централизованного стандарта публикации криптоданных, общее движение к открытости и унификации идёт через использование общепринятых API-спецификаций и унифицированных форматов ответов. Это создаёт условия для построения надёжных систем сбора и анализа данных, обладающих высокой степенью масштабируемости и совместимости между различными источниками.

1.2. Сравнительный анализ сервисов для аналитики крипторынка

В рамках криптовалютной индустрии существует широкий спектр платформ, предоставляющих открытые данные для анализа рыночной динамики, поведения пользователей и технических характеристик блокчейнов. Эти платформы отличаются как по структуре предоставляемых данных, так и по степени открытости, удобству интеграции и надежности. Ниже представлен сравнительный анализ наиболее популярных решений.

Рассмотрим наиболее популярные сервисы, которые предоставляют открытый доступ к рынку.

1.2.1 Агрегатор рыночных данных CoinGeko

Целью данной платформы является предоставление открытых и всесторонних рыночных данных по криптовалютам и связанным проектам. Интерфейс, предоставляющий информацию о наборе данных представлен на рисунке 1.

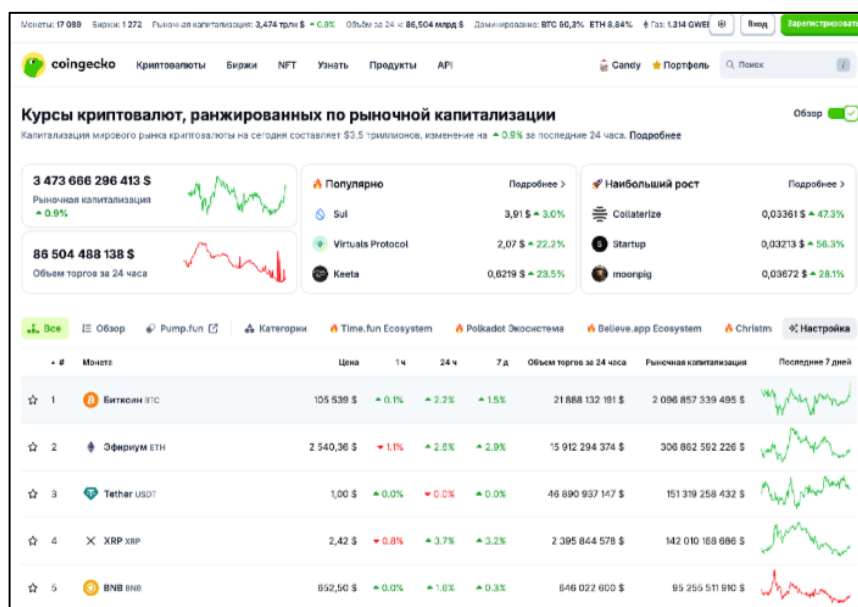


Рисунок 1 – Данные о крипторынке на платформе CoinGeko

На данном ресурсе можно совершать различные действия:

- получить цену, объем торгов, капитализацию монет;
- отследить DeFi, NFT и биржи;
- использовать открытое API без авторизации (ограничения по частоте).

Преимущества:

- бесплатный и довольно открытый API;
- многосторонняя информация: DeFi, деривативы, NFT;
- удобная документация.

Недостатки:

- частота запросов ограничена;
- данные могут обновляться с задержкой.

Отличный выбор для быстрого старта в анализе рынка с минимальными ограничениями. Хорош для статистики и витрин данных. Но из-за ограниченного кол-ва доступных запросов и задержки обновления данных, пользоваться может быть не совсем удобно.

1.2.2 Платформа CoinMarketCap

Цель платформы: премиальный агрегатор рыночной информации с акцентом на качество, проверку и коммерческое использование. На рисунке 2 представлен предварительный просмотр данных рынка.

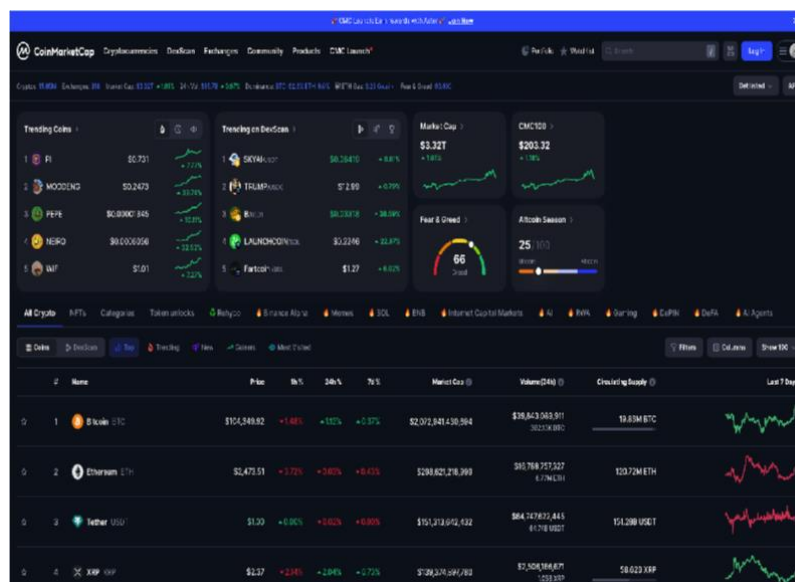


Рисунок 2 – Предварительный просмотр данных CoinMarketCap

На данном ресурсе можно совершать различные действия:

- получить рыночные данные и рейтинги;
- использовать API для отображения на веб-сайтах;
- отслеживать изменения по категориям токенов.

Преимущества:

- высокое доверие в индустрии;
- стандартизированные данные;
- хорошее визуальное представление;
- наличие программного интерфейса REST API в форматах JSON;

Недостатки:

- требуется API-ключ;
- ограничения в бесплатной версии;
- иногда происходят задержки по малым токенам;

Подходит для приложений, где требуется стабильность и высокая достоверность. Хорош для встраивания в публичные панели. Имеет отличную репутацию.

1.2.3 Платформа CryptoCompare

Цель платформы: предоставление комплексной аналитики: рыночной, социальной и исторической информации.

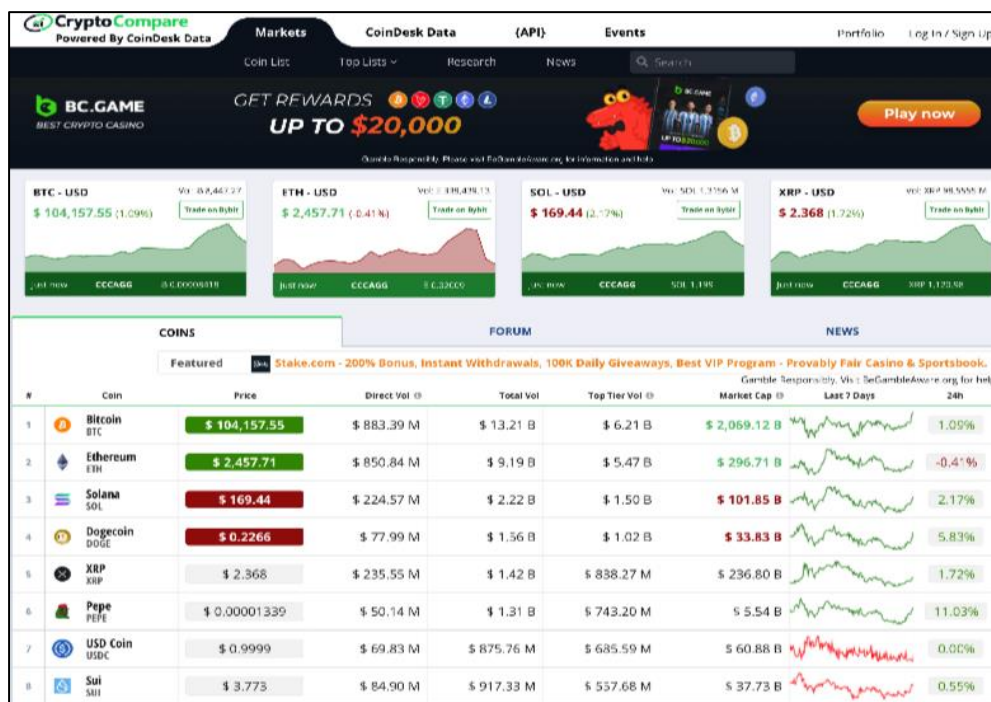


Рисунок 3 – Предварительный просмотр данных CryptoCompare

На данном ресурсе можно совершать различные действия:

- получать исторические и текущие данные;
- строить графики, сравнивать активы;
- анализировать соц. активность.

Преимущества:

- поддержка множества метрик;
- подходит для построения стратегий;
- хорошая документация API;

Недостатки:

- некоторые данные доступны только в платной версии;
- меньшее количество токенов по сравнению с СМС;

Данная платформа является универсальным инструментом для построения торговых и аналитических систем. Особенно полезен для исследователей и трейдеров.

1.2.4 Binance API

Binance – один из крупнейших онлайн-сервисов обмена цифровых валют и блокчейн-система

Целью платформы можно назвать предоставление прямого доступа к данным торгов и пользовательским операциям на крупнейшей криптобирже. На рисунке 4 представлен пример страницы для p2p обмена с другими

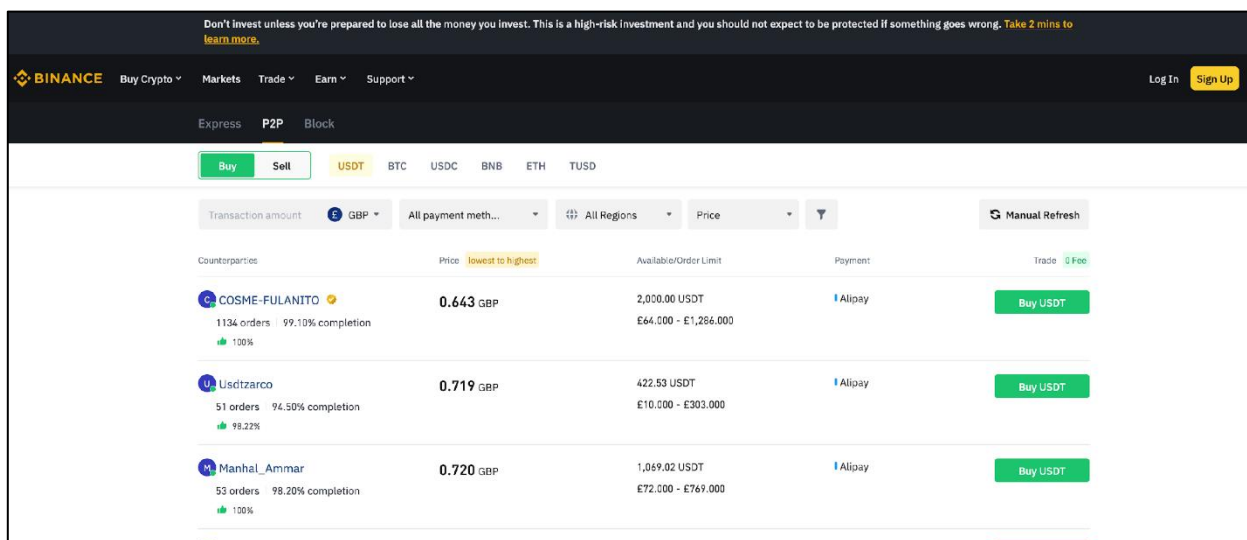


Рисунок 4 – Страница для p2p обмена с другими пользователями биржи

Binance API пользователя биржи.

На данном ресурсе можно совершать различные действия:

- Получать данные стакана и истории сделок;
- Управлять ордерами и счетами;
- Использовать для трейдинговых ботов.

Преимущества:

- реальное время;
- высокая частота обновления;
- поддержка маржи, фьючерсов и спота;

Недостатки:

- привязан к Binance, нет обобщенных данных;
- требуется управление API-ключами;

В целом данный сервис является одной из самых популярных бирж для обмена криптовалютой. Подходит для тех, кто строит трейдинг-алгоритмы и хочет работать с «сырыми» данными. Не подойдет для обзора всего рынка.

1.3 Сравнение сервисов сбора и анализа данных крипторынка

В процессе анализа аналогов было рассмотрено несколько возможных прототипов. Для обоснованного выбора была проведена сравнительная оценка методом анализа иерархий на основе критериев, представленных в Таблице 2.

Таблица 2 – Критерии оценки аналогов

№	Критерий	Обоснование
1	Наличие публичного API	Позволяет интегрироваться и получать данные без необходимости веб-скрейпинга
2	Доступность исторических рыночных данных	Аналитика требует полноты и глубины данных
3	Возможность локального хранения данных	Нужно для анализа без ограничений по количеству запросов или скоростью
4	Наличие базовой аналитики или метрик	Позволяет опираться на уже существующие сценарии анализа
5	Простота реализации и повторяемость	Возможность реализовать функциональность в условиях ВКР

Оцениваем, насколько один критерий важнее другого по 9-бальной шкале Сатти, представленной в Таблице 3.

Таблица 3 – Шкала Сатти

Балл	Значение
1	Равно важны
3	Немного важнее

Продолжение таблицы 3

Балл	Значение
5	Существенно важнее
7	Гораздо важнее
9	Абсолютно важнее
2,4,6,8	Промежуточные значения
x	Если А важнее В, то $B = 1/x$

Затем составляем матрицу парных сравнений, представленную в таблице 4

Таблица 4 – Матрица парных сравнений

Критерий/ Критерий	1	2	3	4	5
1	1	1	3	4	6
2	1	1	3	4	5
3	1/3	1/3	1	2	3
4	1/4	1/4	1/2	1	3
5	1/6	1/5	1/3	1/3	1
Сумма	2.78	2.78	7.83	11.33	18.00

Далее делим каждую ячейку на сумму по колонке и усредняем по строкам, результат нормализации приведен в таблице 5.

Таблица 5 – Вычисление весов(Нормализация)

	1	2	3	4	5	Среднее(Вес)
1	0.36	0.36	0.38	0.35	0.33	0.356

Продолжение таблицы 5

	1	2	3	4	5	Среднее(Вес)
2	0.36	0.36	0.38	0.35	0.28	0.346
3	0.12	0.12	0.13	0.18	0.17	0.144
4	0.09	0.09	0.06	0.09	0.17	0.100
5	0.06	0.07	0.04	0.03	0.06	0.052

Для обоснования весов критериев использован метод анализа иерархий (АНР), позволяющий сравнивать важность каждого критерия относительно других. В результате парных сравнений были получены следующие веса:

Критерий 1(Публичный API) – 35.6%

Критерий 2(Доступ к историческим данным) – 34.6%

Критерий 3(Локальное хранение) – 14.4%

Критерий 4(Наличие аналитики) – 10%

Критерий 5(Простота реализации) – 5.2%

Как видно, наибольшее влияние на выбор прототипа оказывают критерии доступа к данным и возможности интеграции, что соответствует целям построения аналитического сервиса.

Далее следует оценить аналоги по выделенным критериям. Оценку будем проводить по 5-ти бальной шкале, и посчитать итоговую оценку, выбрав прототип для разрабатываемого сервиса, оценка представленная в таблице 6

Таблица 6 – Оценка аналогов по критериям

Критерий/прототип	1	2	3	4	5	Итоговая Оценка
CoinGecko	5*0.356	4*0.346	2*0.144	2*0.1	3*0.052	3.81
CryptoCompare	5*0.356	5*0.346	2*0.144	3*0.1	3*0.052	4.26
CoinMarketCap	5*0.356	3*0.346	2*0.144	2*0.1	2*0.052	3.41
Binance API	5*0.356	5*0.346	4*0.144	4*0.1	5*0.052	4.75

Применением метода анализа иерархий было установлено, что прототипом для разрабатываемой системы наиболее целесообразно выбрать Binance API т.к. его оценка наибольшая и составляет 4.75.

Он удовлетворяет ключевым критериям, таким как наличие открытого API, доступ к историческим данным и возможность построения аналитических сценариев.

1.4 Методы анализа данных криптовалютного рынка

Одним из ключевых этапов разработки аналитических систем является выбор и обоснование методов анализа данных, которые позволяют извлекать значимую информацию из потока рыночных котировок. Для криптовалютного рынка, характеризующегося высокой волатильностью и нестабильностью, особенно актуальны методы, основанные на статистической обработке временных рядов.

В рамках данной работы используются базовые методы математической статистики, такие как расчёт средних значений, экстремумов, относительных изменений, а также агрегирование данных по временным интервалам. Эти методы позволяют выявить:

- амплитуду колебаний цен;
- долю восходящих и нисходящих свечей;
- процентные изменения цены;
- экстремальные значения активов;
- показатели волатильности.

Данные методы служат основой для первичного анализа рыночной активности, построения гипотез о текущих трендах и потенциального применения в более сложных моделях прогнозирования.

1.4.1 Средняя амплитуда свечей

Под амплитудой свечи в контексте японского графика понимается разность между максимальным и минимальным значением цены внутри

одного временного интервала. Вычисление среднего значения амплитуд за период позволяет оценить средний уровень волатильности актива.

Волатильность – ключевой индикатор рыночной нестабильности. Чем выше средняя амплитуда, тем более рискованным считается актив, и наоборот. Этот показатель полезен при формировании торговых стратегий: в условиях высокой волатильности возможны спекулятивные сделки, в то время как низковолатильные активы подходят для долгосрочного инвестирования.

Формула подсчета средней амплитуды свечей по символу и часу [9]:

$$A_{s,t} = \frac{1}{n_{s,t}} \sum_{i=1}^{n_{s,t}} (H_i - L_i),$$

где H_i – значение high для свечи i , L_i – значение low для свечи i , $A_{i,t}$ – средняя амплитуда по символу за час, $n_{s,t}$ – количество свечей по символу в час.

1.4.2 Соотношение бычьих и медвежьих свечей

Анализ вывод о преобладающем тренде: если преобладают бычьи свечи, это может свидетельствовать о растущем интересе инвесторов и повышающемся тренде. Соотношение помогает предсказать разворот тренда и оценить устойчивость текущего движения цены. Формула подсчета бычьих свечей [9]:

$$B_{s,t} = \sum_{i=1}^{n_{s,t}} \delta(C_i > O_i),$$

Формула подсчета медвежьих свечей:

$$M_{s,t} = \sum_{i=1}^{n_{s,t}} \delta(C_i < O_i),$$

где $C_i \in R$ – цена закрытия i -й свечи, $O_i \in R$ – цена открытия i -й свечи, $n_{s,t}$ – количество всех свечей актива в час.

Бычьей свечой считается та, у которой цена закрытия выше цены открытия (рост), медвежьей – наоборот (падение). Подсчёт количества каждого типа свечей и их соотношения позволяет оценить направление рыночных настроений.

1.4.3 Анализ волатильности актива

Волатильность является ключевым индикатором риска и изменчивости актива. В контексте анализа волатильность актива выражается как средний диапазон изменения цены. В целом волатильность отвечает на вопрос: «Насколько актив стабилен?». Формула вычисления средней волатильности имеет вид [9]:

$$V_{avg} = \frac{1}{n} \sum_{i=1}^n (H_i - L_i).$$

1.4.4 Процент изменения цены

Процентное изменение цены рассчитывается как разность между текущей и предыдущей ценой закрытия, делённая на предыдущую цену, выраженная в процентах. Этот показатель позволяет количественно оценить темп изменения цены. Этот показатель является основой для многих технических индикаторов и используется при вычислении доходности, определения волатильности и идентификации краткосрочных аномалий в движении цены. Показатель процента изменения цены вычисляется по формуле [9]:

$$\Delta P_i = \left(\frac{C_i - C_{i-1}}{C_{i-1}} \right) \cdot 100,$$

где $C_i \in R$ – цена закрытия i -й свечи, $C_{i-1} \in R$ – цена закрытия i -й-1 свечи, ΔP_i – процентное изменение цены закрытия.

1.4.5 Экстремальные значения цены

Анализ экстремального значения цены заключается в вычислении максимального и минимального значения цены актива за заданную дату.

Знание краткосрочных экстремумов позволяет определить уровни сопротивления и поддержки – важнейшие технические уровни в трейдинге.

1.5 Постановка задачи

Основная задача сервиса для сбора данных и анализа рынка криптовалют состоит в том, чтобы предоставить основным ее пользователям средства для свободного сбора необходимой информации и аналитики, а также предоставить инструментарий для просмотра аналитических данных с целью выявления ключевых рыночных показателей и предоставления аналитической информации пользователю.

С развитием цифровых финансовых инструментов рынок криптовалют стал важной частью глобальной экономики. Он характеризуется высокой динамичностью, волатильностью и доступностью большого объема открытых данных. Для участников рынка – трейдеров, инвесторов, аналитиков – крайне важна возможность оперативно получать и анализировать данные, чтобы принимать обоснованные решения. Однако вручную следить за изменениями рынка сложно и неэффективно. Поэтому возникает необходимость в создании автоматизированной системы, способной собирать и анализировать данные крипторынка.

Таким образом, разрабатываемое приложение должно решать следующие задачи:

- автоматизированный сбор данных, получение исторических и текущих данных по торговым свечам, стаканам ордеров, позициям, ценам и другим метрикам, поддерживая параметризованные запросы данных;
- передача и буферизация данных, использование Apache Kafka как брокера сообщений для передачи сырых данных от компонента сбора к компоненту хранения и аналитики;
- хранение и агрегация данных, сохранение данных в ClickHouse – аналитическую колоночную СУБД, оптимизированную под обработку больших объёмов временных рядов;
- анализ и расчёт метрик: вычисление показателей волатильности, амплитуды, доли «бычьих» и «медвежьих» свечей; построение статистики

изменения цен в процентах; получение максимумов/минимумов цен за заданный период; выявление наиболее стабильного актива;

- разрабатываемая информационная система должна включать интерфейс для запуска анализа и отображения результатов, ориентированный на конечного пользователя.

Разрабатываемая система должна быть удобна в эксплуатации, уметь обрабатывать данные поставщика информации о рынке и предоставить пользовательский интерфейс для взаимодействия с системой.

1.6. Анализ требований

Анализ требований является важным этапом проектирования, на котором в соответствии с бизнес-процессами можно сформулировать функциональные требования к системе.

Проектируемый сервис предназначен для сбора и анализа данных с открытого API биржи, а затем предоставлением аналитической информации по этим данным пользователю в простом графическом виде, являющимся таблицей.

Каждый авторизованный пользователь должен иметь доступ к проведению каждого из доступных анализов, а также доступ к запросу информации с открытого API биржи с нужными пользователю параметрами, которые затем добавятся в общую выборку для анализа

1.6.1 Функциональные требования

Согласно поставленным задачам, были выделены следующие функциональные требования к проектируемой системе:

- регистрация и аутентификация;
- поддержка конфигурации (символ, категория, интервал, лимит и т.д.);
- запуск аналитических запросов;

- возможность посылать запросы о необходимых данных напрямую поставщику информации о рынке через сервис
- обработка ошибок и стабильность;
- просмотр аналитических данных, полученных из запроса;
- предоставление интерфейса взаимодействия с веб-сервисом

Диаграмма вариантов использования сервиса представлена на рисунке 8

представлена.

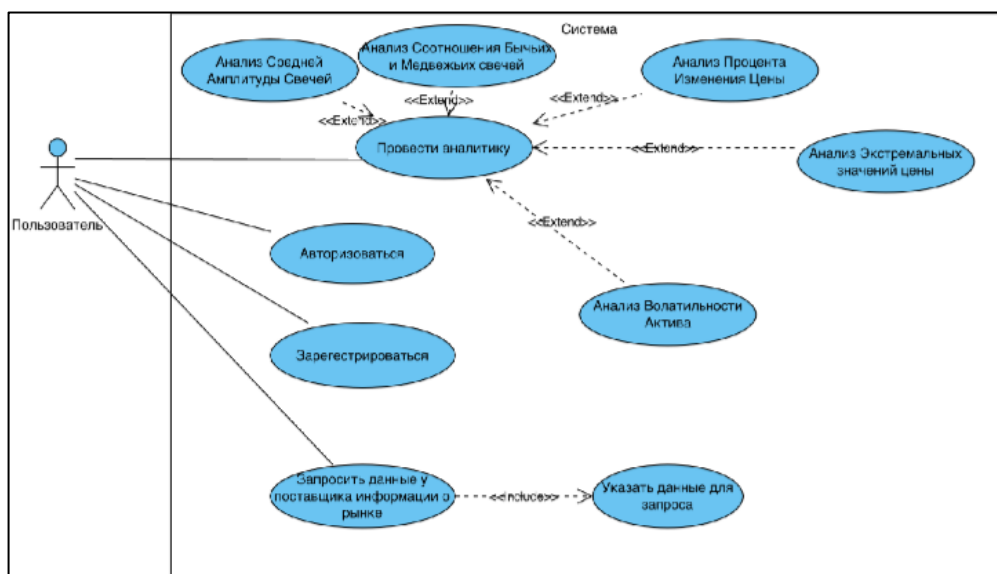


Рисунок 8 – Диаграмма вариантов использования сервиса

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы при выполнении определенных функций.

На рисунке 9 приведены спецификации варианта использования «Зарегистрироваться».

Прецедент: зарегистрироваться
ID: 1
Краткое описание: пользователь регистрируется в системе
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: нет
Основной поток: 1. Пользователь вводит некорректные данные (некорректные символы, почта, длина пароля) 1.1. Система выводит предупреждение о некорректном вводе 2. Пользователь с введенной почтой существует 2.2. Система выводит предупреждение об уже существующей почте 3. Пользователь добавляется в систему
Постусловие: пользователь добавлен в базу данных
Альтернативные потоки: нет

Рисунок 9 – Сценарий выполнения варианта использования
«Зарегистрироваться»

На рисунке 10 приведены спецификации варианта использования «Запросить данные у поставщика».

Прецедент: запросить данные у поставщика
ID: 2
Краткое описание: пользователь выбирает тип запроса и вводит его параметры, затем получает информацию по сформированному запросу от поставщика
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: пользователь авторизован
Основной поток: 1. Пользователь выбирает тип запроса и вводит параметры запроса 1.1. Система выводит делает запрос поставщику 1.2. Система записывает данные в базу 1.3. Система отображает данные пользователю 2. Пользователь может просмотреть полученные данные
Постусловие: пользователю отображены данные по его запросу, и данные этого запроса сохранены в базу
Альтернативные потоки: нет

Рисунок 10 – Сценарий выполнения варианта использования «Запросить
данные у поставщика»

На рисунке 11 приведены спецификации варианта использования «Провести аналитику».

Прецедент: провести аналитику
ID: 3
Краткое описание: пользователь выбирает необходимый ему анализ, и совершает его на данных, собранных за определенный период
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: пользователь авторизован
Основной поток: 1. Пользователь выбирает анализ и вводит необходимый период 1.1. Система отображает таблицу с аналитическими данными
Постусловие: отображены данные
Альтернативные потоки: нет

Рисунок 11 – Сценарий выполнения варианта использования «Провести аналитику»

1.6.2 Нефункциональные требования

Серверная часть приложения должна быть выполнена на языке Golang. Язык Golang был выбран так, как он обладает высокой производительностью, что необходимо при работе с потоками данных и при выполнении аналитических операций, быстрой компиляцией и развертыванием, низким потреблением памяти, и наличием библиотек для работы с Kafka, Clickhouse, PostgreSQL. Также имеет поддержку многопоточности через горуты [11], что критично при потреблении данных из Kafka и обработке большого количества входящих HTTP-запросов.

Система должна предоставлять RESTful API, реализующее все функции по принципу клиент-серверного взаимодействия. Это обеспечивает простую интеграцию с фронтендом (веб-приложение на React), а также потенциальную возможность использовать сервис сторонними клиентами (например, мобильными приложениями).

Потоковая обработка данных: Kafka, используемая как брокер сообщений [21] для обеспечения надежной очереди, гарантирующей доставку сообщений, которые мы будем получать от поставщика данных, и затем собирать и хранить в Clickhouse, с которым она легко интегрируется.

Аналитика и хранение большого объема данных: ClickHouse, т.к. Это колоночная аналитическая база данных [19], идеально подходящая для хранения и анализа временных рядов и событий, что обеспечивает высокую скорость выполнения сложных запросов на больших объемах данных, используется как основное хранилище торговых данных.

В качестве транзакционной СУБД будет использоваться PostgreSQL для хранения данных о пользователях и авторизации.

Клиентская часть приложения будет реализована с использованием JavaScript и React. Предусматривается поддержка работы в современных веб-браузерах. При разработке необходимо использовать среду разработки Goland, а также редактор кода Visual Studio Code, для развертывания приложения на сервере необходимо использовать Docker.

2. Проектирование сервиса сбора и анализа данных крипторынка

2.1 Архитектура сервиса

Архитектура сервиса построена по принципам микросервисной архитектуры с выделением отдельных компонентов для авторизации, аналитики, хранения данных и взаимодействия с внешними источниками. Это позволяет масштабировать части приложения независимо друг от друга, повышать упростить сопровождение и отказоустойчивость. На рисунке 12 представлена схема взаимодействия сервиса с REST API.

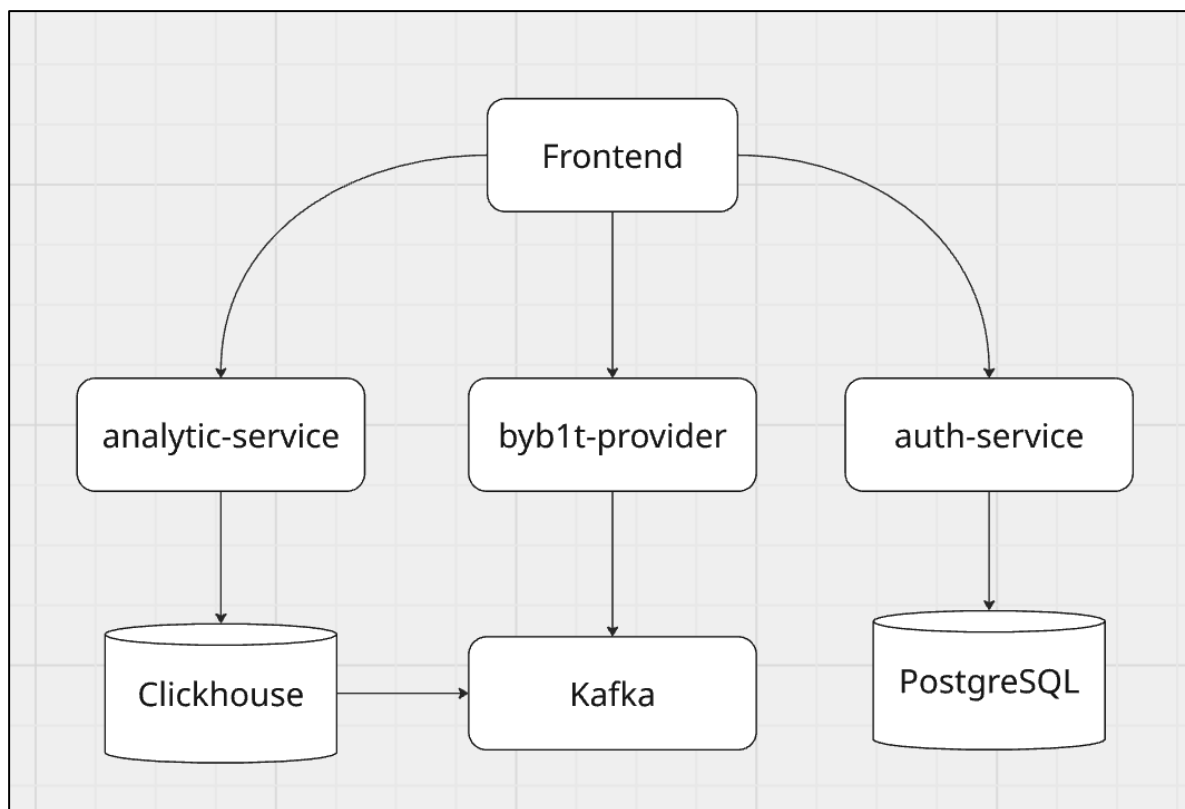


Рисунок 12 – Схема взаимодействий приложений с API в микросервисной архитектуре

Из схемы взаимодействия видно, что данная архитектурная модель системы представляет собой событийно-ориентированную архитектуру т.к. взаимодействие основных компонентов строится через обмен событиями.

Преимущества, которая дает данная модель:

- асинхронность: отправка и обработка событий не блокируют друг друга;
- масштабируемость: каждый сервис масштабируется независимо, а также есть возможность горизонтального масштабирования Kafka и Clickhouse;
- гибкость: можно разрабатывать каждый сервис отдельно;
- устойчивость к сбоям т.к. Kafka сохраняет сообщения - данные не теряются;
- высокая производительность хранения и аналитики;
- логичное разделение ответственности;
- возможность внедрения новых источников данных без изменения существующих сервисов.

Ограничения и сложности:

- сложность сопровождения т.к. микросервисы требуют инфраструктуры;
- сложность отладки;
- временная неактуальность данных вследствие не мгновенного попадания данных из Kafka в Clickhouse.

2.2 Проектирование структур данных

2.2.1 Логическая модель базы данных аутентификации

На основе проведенного анализа функциональных требований и предметной области была разработана логическая модель базы данных аутентификации.

В начале необходимо выделить список всех основных сущностей, использующихся в процессе аутентификации. Данный список сущностей базы данных представлен в таблице 2.

Таблица 7 – Описание сущностей базы данных аутентификации

№	Наименование	Описание
1	Пользователь	Информация о пользователях, зарегистрированных в системе
2	Токен обновления	Список токенов, использующихся для актуализации аутентификации пользователей

На основании данных сущностей необходимо построить модель. Для создания модели использовалось приложение Visual Paradigm. На рисунке 13 изображена построенная логическая модель сервиса.

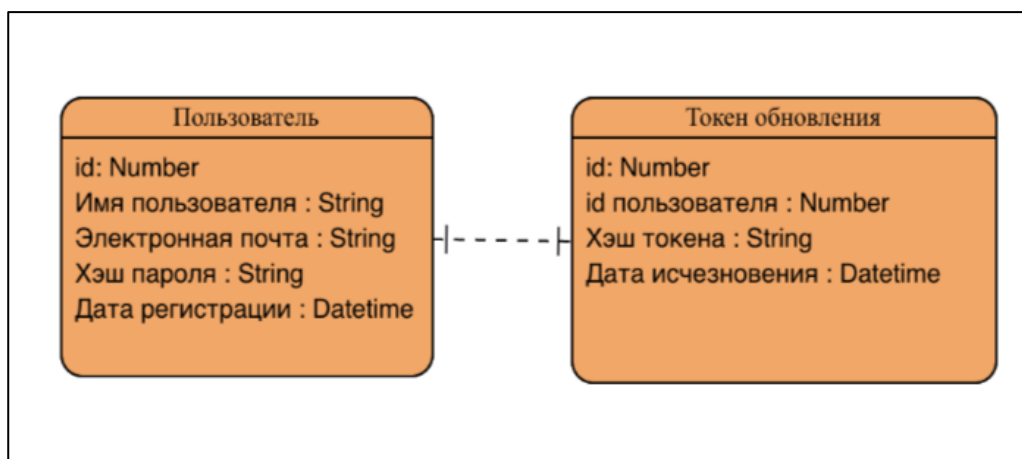


Рисунок 13 – Логическая схема БД аутентификации

От данного уровня модели БД нужно перейти к физическому.

2.2.2 Физическая модель базы данных аутентификации

Физическая модель – логическая модель базы данных, выраженная через диалект языка описания конкретной СУБД. Физические модели баз данных определяют способы размещения данных, которые поддерживаются на физическом уровне. В качестве СУБД будет использован PostgreSQL.

Плюсом данной СУБД является то, что она некоммерческая. Также данная СУБД имеет поддержку множества различных типов данных, такие как JSON, XML, массивы, uuid идентификаторы и т.д.

На основании логической модели была разработана физическая модель базы данных. Физическая модель БД аутентификации представлена на рисунке 14.

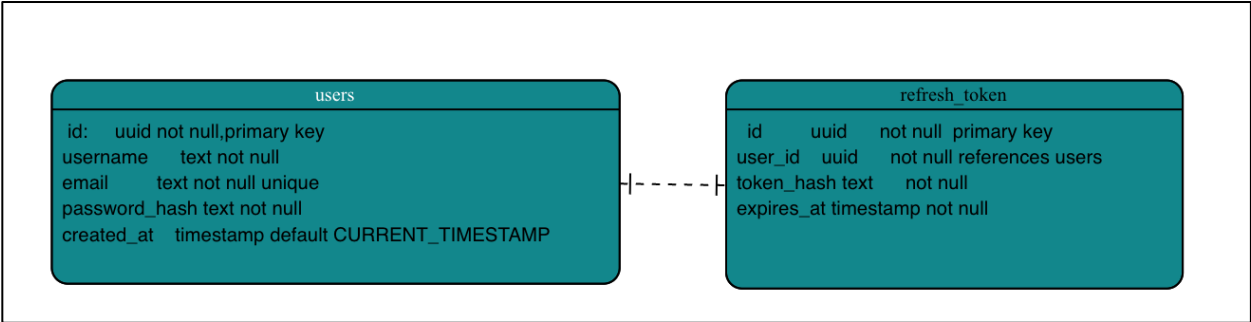


Рисунок 14 – Физическая схема БД аутентификации

В процессе создания физической модели было принято решение использовать для создания модели БД миграции.

2.2.3 Проектирование таблиц под аналитические запросы

Также на основе проведенного анализа предметной области и функциональных требований была спроектирована схема базы данных Clickhouse для хранения и анализа данных, полученных от поставщика информации о рынке криптовалют. В качестве базы данных выбрана Clickhouse, так как является высокопроизводительной системой управления и хранения аналитических данных, оптимизированной для обработки больших объемов данных в реальном времени. Выбор ClickHouse обусловлен её способностью эффективно справляться с большими наборами данных, а также высокой скоростью выполнения запросов, что критично для аналитических задач, таких как расчёт процентных изменений цен и волатильности. Кроме того, ClickHouse обеспечивает масштабируемость и возможность партиционирования данных по времени, что позволяет оптимизировать хранение исторических данных и ускорить доступ к ним.

Описание сущностей, используемых в процессе анализа данных, представлено в таблице 8.

Таблица 8 – Описание сущностей базы данных аналитики

Наименование	Назначение
Ставка фондирования	Информация о ставках фондирования
Свеча	Информация о котировках цен для различных торговых пар
Монета	Информация о криптовалютах и метаданных о монетах
Цена доставки	Информация о ценах доставки фьючерсных контрактов
Стакан заявок	Информация о заявках на покупку и продажу для торговых пар
Тикер	Сущность представляет агрегированные данные о текущем состоянии рынка для торговых пар
Позиция	Сущность хранит информацию о текущих открытых позициях пользователя на платформе

Схема базы данных представлена на рисунке 15

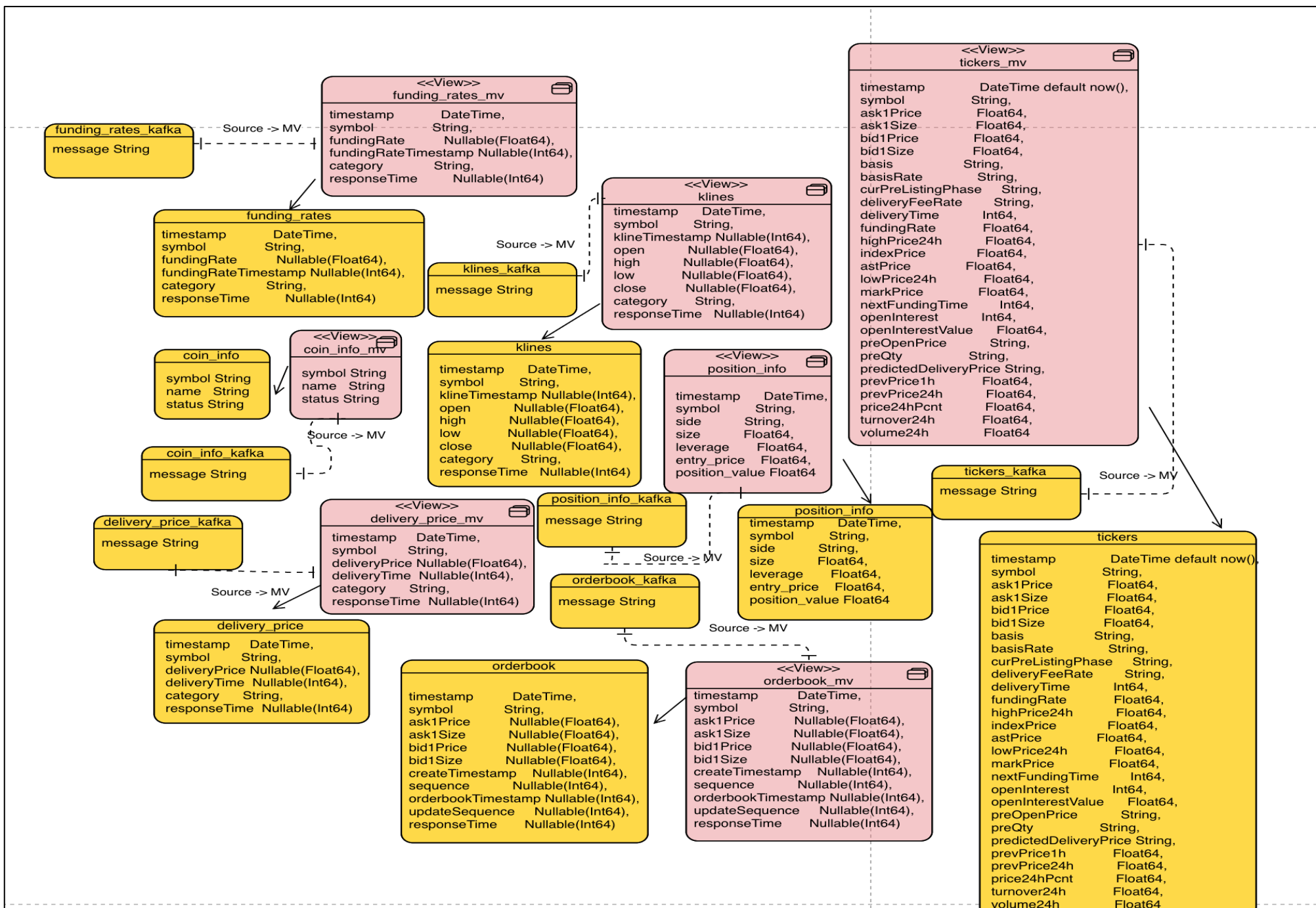


Рисунок 15 - Схема Clickhouse

2.2.4. Проектирование топиков Kafka

Для обеспечения эффективной обработки данных в реальном времени, в рамках архитектуры приложения была спроектирована система топиков Apache Kafka. Kafka выбрана как распределённая платформа потоковой обработки благодаря её высокой пропускной способности, устойчивости к сбоям и способности масштабироваться под растущие объёмы данных.

Список топиков:

- positionInfo – Хранит информацию о ставках позициях;
- orderBook – Хранит информацию о стаканах заявок;
- fundingRate – Хранит информацию о ставках фондирования;
- deliveryPrice – Хранит информацию о ценах доставки;
- coinInfo – Хранит информацию о монетах;
- tickers – Хранит информацию о тикерах;
- klines – Хранит информацию о свечах.

2.3 Проектирование программных средств

2.3.1 Проектирования сервиса работы с API биржи

Byb1t-provider предоставляет функциональность для взаимодействия с API byb1t. Сервис пушит сообщения в брокер сообщений, из которого затем будет читать Clickhouse

Для взаимодействия фронтенда с сервером в сервисе разработан API, описание которых приведено в таблице 4.

Таблица 9 – byb It-provider API

URL запроса	Http метод	Описание	Query параметры запроса
/market/position/info	GET	Информация о позиции в реальном времени	category, symbol, baseCoin, settleCoin, limit, cursor
/market/orderbook	GET	Запрос на глубину журнала выкупа	category, symbol, limit

Продолжение таблицы 9

URL запроса	Http метод	Описание	Query параметры запроса
/market/funding-rate	GET	История ставок финансирования	category, symbol, limit, startTime, endTime
/market/delivery-price	GET	Цена доставки	category, symbol, limit, baseCoin, cursor
/market/tickers	GET	Актуальный снимок цен	category, symbol, baseCoin, expDate
/market/coin-info	GET	Информация о монете	coin

Диаграмма классов представлена на рисунке 16

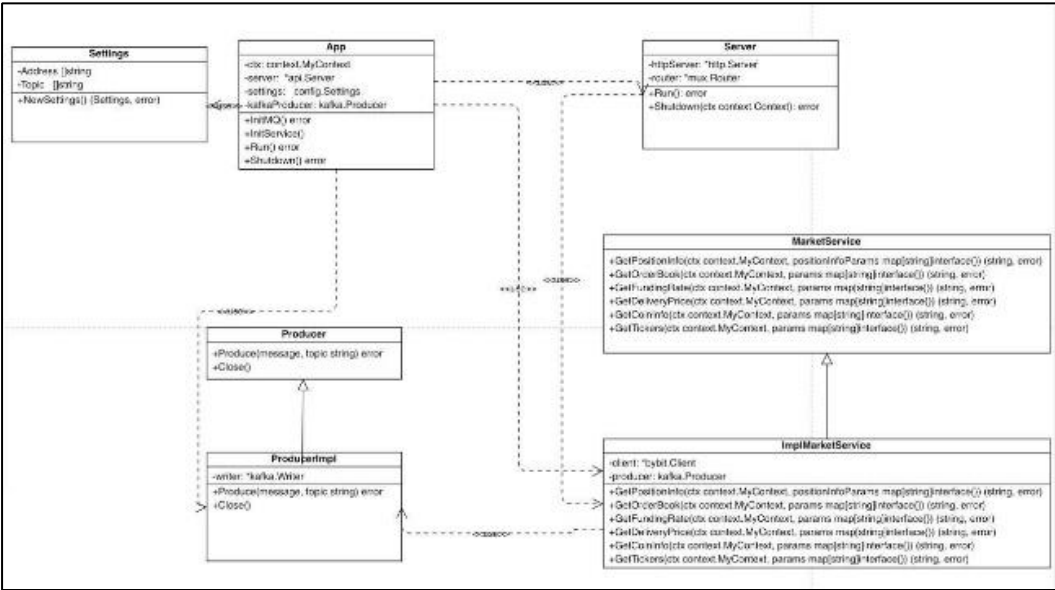


Рисунок 16 – Диаграмма классов bybit-provider

Описание классов на рисунке 16 представлено ниже:

Назначение сервиса – обработка и запись в очередь запросов, направляемых в API биржи.

Описание методов классов сервиса bybit-provider:

Методы класса ImplMarketService:

- GetPositionInfo(ctx context.MyContext, positionInfoParams map[string]interface{}) (string, error): получение информации по позиции;
- GetOrderBook(ctx context.MyContext, params map[string]interface{}) (string, error): получить список ордеров;
- GetFundingRate(ctx context.MyContext, params map[string]interface{}) (string, error): Получение фандинга во фьючерсах;

- GetDeliveryPrice(ctx context.MyContext, params map[string]interface{ }) (string, error): Получение стоимости отправки монеты;
- GetCoinInfo(ctx context.MyContext, params map[string]interface{ }) (string, error): Получение информации о монете;
- GetTickers(ctx context.MyContext, params map[string]interface{ }) (string, error): Получение тикеров.

Описание ProducerImpl:

Назначение: отвечает за работу с записью в очередь.

Методы класса:

- Produce(message, topic string): Пишет сообщение в очередь.
- Close(): закрывает запись.

2.3.2 Проектирование сервиса Аналитики

Analytic-service предоставляет функциональность для аналитики данных криптобиржи.

Для хранения данных, которые нужно проанализировать сервис использует Clickhouse.

Для взаимодействия фронтенда с сервером в сервисе разработан API, описание которых приведено в таблице 10.

Таблица 10 – analytic-service API

URL запроса	Http метод	Описание	Query параметры запроса
/analytics/avg-amplitude	GET	Анализирует среднюю амплитуду свечей	date
/analytics/volatility	GET	Анализирует среднюю волатильность свечей	date
/analytics/bullish-bearish-ratio	GET	Анализирует соотношение быков к медведям	date
/analytics/price-extremes	GET	Анализирует максимумы и минимумы цен на позиции	date

Продолжение таблицы 10

URL запроса	Http метод	Описание	Query параметры запроса
/analytics/price-change-percent	GET	Анализирует процент изменения цен на позиции	date

Диаграмма классов представлена на рисунке 17

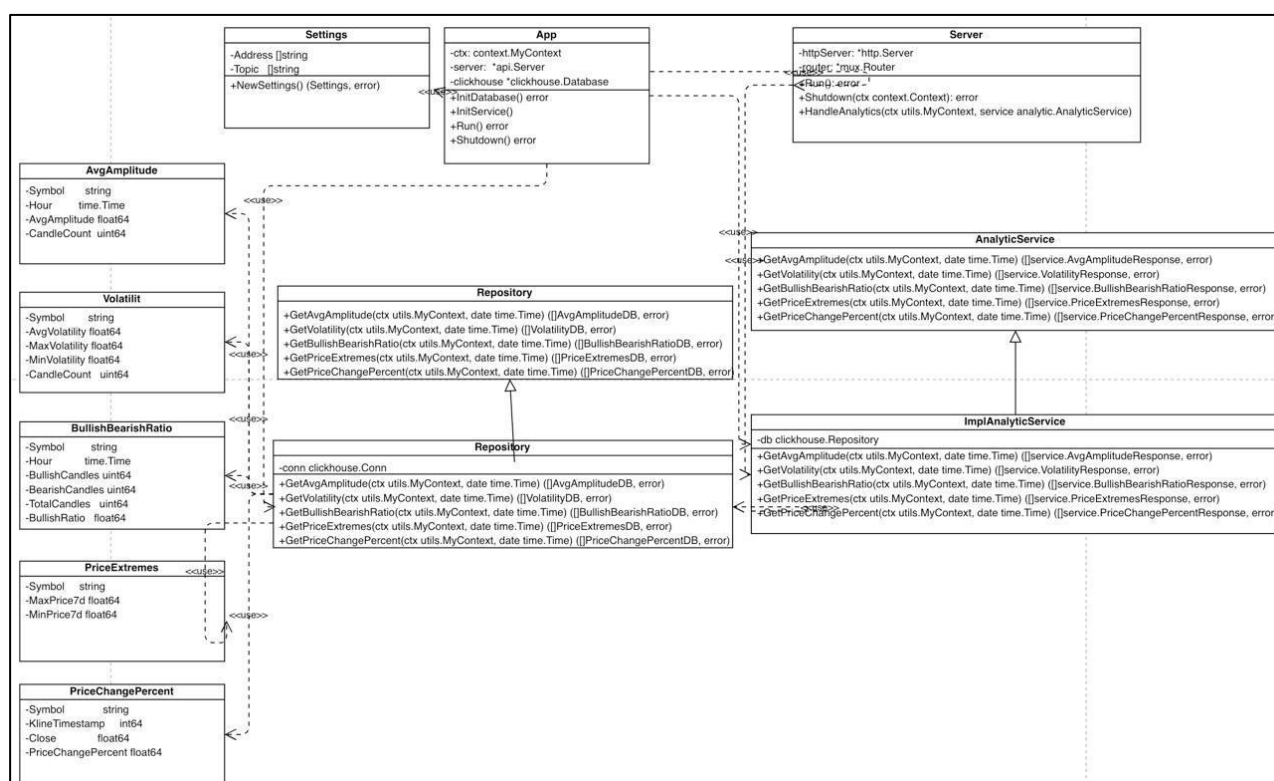


Рисунок 17 – Диаграмма классов Analytic-service

Назначение: структуры для работы с данными сервиса.

Описание ImplAnalyticService:

Назначение: отвечает за обработку запросов связанных с аналитикой данных.

Методы класса:

- GetAvgAmplitude(ctx utils.MyContext, date time.Time):
Анализирует среднюю амплитуду свечей;
- GetVolatility(ctx utils.MyContext, date time.Time):
Анализирует среднюю волатильность свечей;
- GetBullishBearishRatio(ctx utils.MyContext, date time.Time):
Анализирует соотношение быков к медведям;

- `GetPriceExtremes(ctx utils.MyContext, date time.Time):`

Анализирует максимумы и минимумы цен на позиции;

- `GetPriceChangePercent(ctx utils.MyContext, date time.Time):`

Анализирует процент изменения цен на позиции.

Описание `AnalyticClickhouse`:

Назначение: отвечает за работу с базой данных.

Методы класса:

- `GetAvgAmplitude(ctx utils.MyContext, date time.Time):` Получет

аналитику из бд по параметрам пользователя;

- `GetVolatility(ctx utils.MyContext, date time.Time):` Получет

аналитику из бд по параметрам пользователя;

- `GetBullishBearishRatio(ctx utils.MyContext, date time.Time):`

Получет аналитику из бд по параметрам пользователя;

- `GetPriceExtremes(ctx utils.MyContext, date time.Time):` Получет

аналитику из бд по параметрам пользователя;

- `GetPriceChangePercent(ctx utils.MyContext, date time.Time):`

Получает аналитику из бд по параметрам пользователя.

2.3.3 Проектирование сервиса аутентификации

`Auth Service` предоставляет функциональность для управления учетными записями пользователей, включая регистрацию и авторизацию и обновление токенов. Сервис обеспечивает возможность создания новых учетных записей, а также входа в сервис по почте и паролю.

Для хранения данных о пользователях сервис использует базу данных, в которой хранятся данные пользователя и рефреш-токены.

Для взаимодействия фронтенда с сервером в сервисе разработан API, описание которых приведено в таблице 11.

Таблица 11 – Auth Service API

URL запроса	Http метод	Описание	Query параметры запроса
api/register/	POST	Регистрация пользователя	email, username, password
api/login/	POST	Авторизация пользователя	email, password
api/refresh/	POST	Обновление токенов	accessToken, refreshToken

Диаграмма классов представлена на рисунке 18

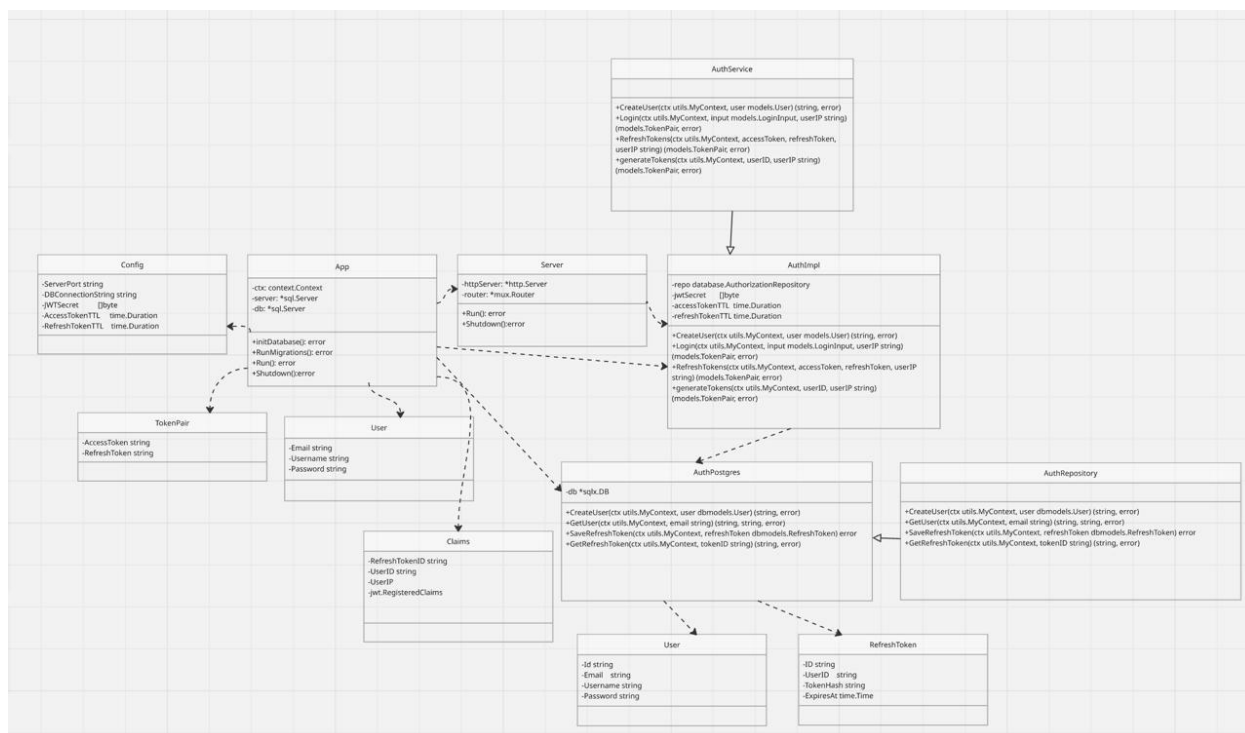


Рисунок 18 – Диаграмма классов Auth Service

Описание структур User, LoginInput, TokenPair, Claims, RefreshToken:

– Представляют собой модели данных, используемые для взаимодействия с сервисами системы.

Описание MyContext:

– Служит для централизованного хранения контекста приложения и логгера, упрощая их передачу между различными компонентами системы.

Методы:

- `NewMyContext(ctx context.Context, logger *zap.SugaredLogger)`

`MyContext`: создаёт новый экземпляр `MyContext`, принимая контекст и логгер в качестве параметров.

Описание `Config`:

- Предназначен для хранения настроек приложения, включая параметры сервера, баз данных, JWT и срок действия токенов.

Методы:

- `NewConfig() (Config, error)`: формирует новый объект конфигурации, извлекая данные из переменных окружения, проверяя их корректность и подготавливая значения для работы приложения.

Описание `App`:

- Управляет жизненным циклом приложения, включая настройку контекста, подключение к базе данных, запуск HTTP-сервера, выполнение миграций, инициализацию сервисов и корректное завершение работы.

Методы:

- `NewApp(ctx context.Context, logger *zap.SugaredLogger, config config.Config) *App`: создаёт новый экземпляр приложения, инициализируя его с заданным контекстом и конфигурацией;
- `InitDatabase() error`: устанавливает соединение с базой данных, реализуя повторные попытки в случае сбоев;
- `RunMigrations() error`: выполняет миграции базы данных с использованием инструмента `goose`;
- `InitService()`: настраивает сервисы и регистрирует маршруты для HTTP-сервера;
- `Run() error`: запускает сервер для обработки запросов в отдельной горутине;
- `Shutdown() error`: останавливает сервер и закрывает соединение с базой данных.

Описание `Server`:

- Отвечает за настройку и управление HTTP-сервером, обработку входящих запросов через маршрутизатор и регистрацию маршрутов. Обеспечивает поддержку CORS, обработку ошибок на уровне сервера и интеграцию с сервисами, такими как авторизация.

Методы:

- `NewServer(ctx utils.MyContext, config config.Config) *Server`: создаёт экземпляр сервера, настраивая маршрутизатор, CORS и промежуточное ПО;
- `Run() error`: активирует HTTP-сервер для обработки запросов;
- `Shutdown(ctx utils.MyContext) error`: завершает работу сервера и освобождает занятые ресурсы;
- `HandleAuth(ctx utils.MyContext, service service.AuthorizationService)`: задаёт маршруты для функций авторизации (регистрация, вход, обновление токенов).

Описание `ImplAuthorizationService`:

- Реализует бизнес-логику, связанную с регистрацией, авторизацией и обновлением токенов, а также передачу данных на уровень репозитория.

Методы:

- `CreateUser(ctx Utils.MyContext, user models.User)`: регистрирует нового пользователя, возвращая его ID или ошибку при неудаче;
- `GetUser(ctx Utils.MyContext, input models.LoginInput, userIP string)`: выполняет авторизацию, возвращая пару токенов или ошибку;
- `RefreshTokens(ctx Utils.MyContext, accessToken, refreshToken, userIP string)`: обновляет токены, возвращая новую пару или ошибку.
- Описание `AuthorizationPostgres`:
- Обеспечивает взаимодействие с базой данных для операций авторизации.

Методы:

- `CreateUser(ctx Utils.MyContext, user dbmodels.User)`: сохраняет данные пользователя в базе;
- `GetUser(ctx Utils.MyContext, email string)`: извлекает информацию о пользователе по email;
- `SaveRefreshToken(ctx Utils.MyContext, refreshToken dbmodels.RefreshToken)`: записывает хэш токена, ID пользователя и срок действия в базу данных;
- `GetRefreshToken(ctx Utils.MyContext, tokenID string)`: получает данные о токене из базы.

2.4 Проектирование пользовательского интерфейса web-приложения

На основе диаграммы вариантов использования, для клиентского приложения был спроектирован графический интерфейс. Пользовательский интерфейс должен обеспечивать простое и эффективное выполнение основных функций пользователей.

Интерфейс пользователя сервиса сбора и анализа данных крипторынка будет состоять из нескольких окон:

- окно авторизации и регистрации;
- окно сбора данных с API криптобиржи
- окно анализа средней амплитуды свечей;
- окно анализа волатильности;
- окно анализа соотношения бычьих и медвежьих свечей;
- окно анализа экстремумов цен;
- окно анализа изменения цен в процентах.

Окно авторизации и регистрации, где реализована форма регистрации и входа в систему

1. Поля для ввода:

- электронная почта;
- имя пользователя;

– пароль.

2. Кнопки:

– регистрация;

– вход.

Макет продемонстрирован на рисунке 19



The image shows a web form titled "Login / Register". It contains three input fields: "email", "username", and "Введите пароль" (Enter password). Below the fields are two buttons: "Register" and "Login".

Рисунок 19 – Окно авторизации и регистрации

Окно сбора данных с API криптобиржи:

1. Поля для ввода:

– эндпоинт;

– параметры запроса.

2. Кнопки:

– отправить запрос.

Макет продемонстрирован на рисунке 20.

Data Collector Interaction



The image shows a form titled "Data Collector Interaction". It has two input fields: "Endpoint" with the value "Candle History (/history/candle)" and "Query Parameters (e.g., category=option&symbol=BTCUSD)". Below the fields is a text label "Required parameters: symbol, interval (e.g., symbol=BTCUSD&interval=60)". At the bottom is a green button labeled "SEND REQUEST".

Рисунок 20 – Окно сбора данных с API криптобиржи

Окно для аналитики средней амплитуды свечей

1. Поля для ввода:

– выбор периода.

Макет продемонстрирован на рисунке 21.

Analytic Dashboard				
DATA COLLECT AVG AMPLITUDE VOLATILITY BULL				
Average Amplitude				
Select Date				
04.04.2025				
Symbol	Hour	Average Amplitude	Candle Count	
BTCUSD	2025-04-04T00:00:00Z	4,831.18	1	
BTCUSD	2025-04-04T01:00:00Z	7,294.32	1	
BTCUSD	2025-04-04T02:00:00Z	5,051.78	1	
BTCUSD	2025-04-04T03:00:00Z	5,024.69	1	
BTCUSD	2025-04-04T04:00:00Z	5,828.71	1	
BTCUSD	2025-04-04T05:00:00Z	6,250.74	1	

Рисунок 21 – Окно для аналитики средней амплитуды свечей

Окно для аналитики волатильности

1. Поля для ввода:

– выбор периода.

Макет продемонстрирован на рисунке 22

Analytic Dashboard				
DATA COLLECT AVG AMPLITUDE VOLATILITY BULLISH/BEARISH RATIO PRICE				
Volatility				
Select Date				
18.03.2025				
Symbol	Average Volatility	Max Volatility	Min Volatility	Candle Count
BTCUSD	6,003.502	12,184.7	3,400.82	201

Рисунок 22 – Окно для аналитики волатильности

Окно для аналитики экстремумов цен

- 1. Поля для ввода:
- выбор периода.

Макет продемонстрирован на рисунке 23

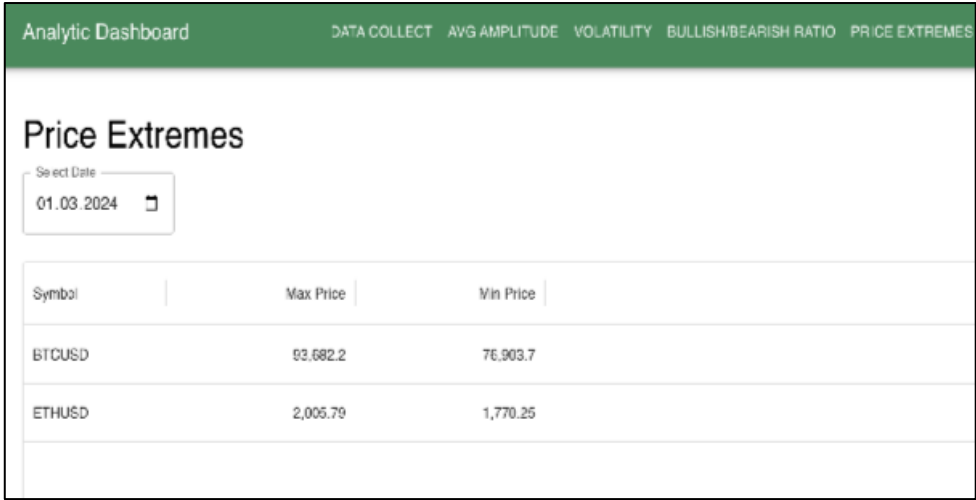


Рисунок 23 – Окно для аналитики экстремумов цен

Окно для аналитики процента изменения цен.

- 1. Поля для ввода:
- выбор периода.

Макет продемонстрирован на рисунке 24.

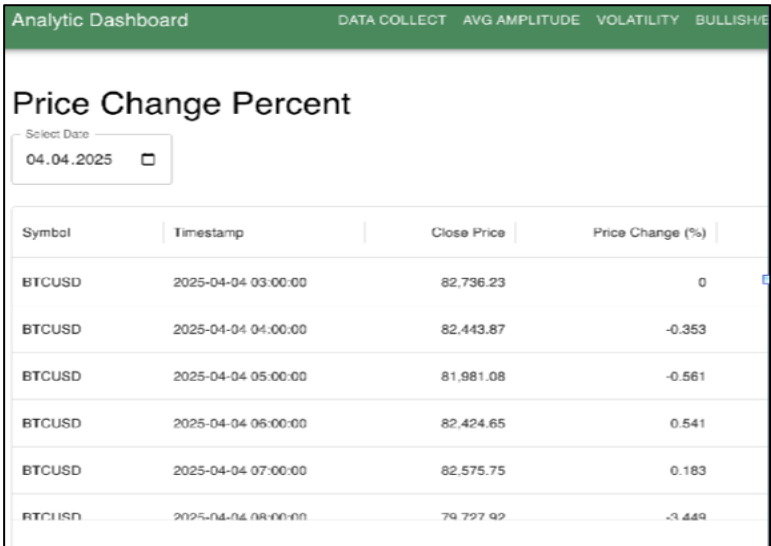


Рисунок 24 – Окно для аналитики процента изменения цен

Окно для аналитики процента изменения цен

- 1. Поля для ввода:

– выбор периода.

Макет продемонстрирован на рисунке 25.

Analytic Dashboard DATA COLLECT AVG AMPLITUDE VOLATILITY BULLISH/BEARISH RATIO PRICE EXTREMES

Bullish/Bearish Ratio

Select Date: 18.03.2025

Symbol	Hour	Bullish Candles	Bearish Candles	Total Candles	Bullish Ratio
BTCUSD	2025-03-27T00:00:00Z	0	1	1	0
BTCUSD	2025-03-27T01:00:00Z	0	1	1	0
RTCUSD	2025-03-27T02:00:00Z	1	0	1	1
BTCUSD	2025-03-27T03:00:00Z	1	0	1	1
BTCUSD	2025-03-27T04:00:00Z	0	1	1	0
RTCUSD	2025-03-27T05:00:00Z	0	1	1	0

1 row selected Rows per page

Рисунок 25 – Окно анализа соотношения бычьих и медвежьих свечей

Любая из страниц содержит в себе React компоненты, и использует библиотеку axios.

3. Реализация программного обеспечения сервиса

Программные средства сервиса сбора и аналитики данных крипторынка.

Компоненты данного сервиса представлены на рисунке 26.

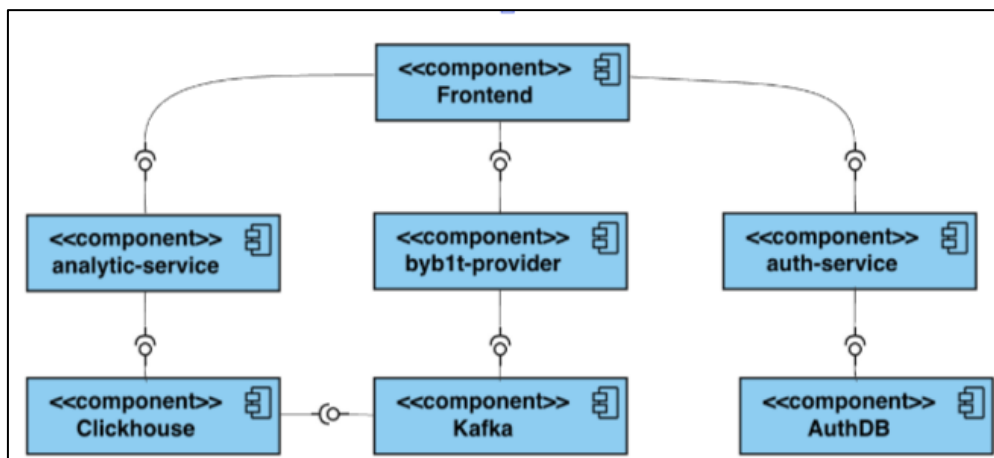


Рисунок 26 – Диаграмма компонентов

Таблица 12 – Описание компонентов системы

№	Название	Описание
1	Frontend	Веб-клиент на React.js. Интерфейс пользователя. Позволяет авторизоваться, запускать анализ данных и просматривать результаты. Взаимодействует с: Byb1t-provider, Auth-service, Analytic-service
2	Byb1t-provider	Сервис, который подключается к публичному API Bybit, получает рыночные данные (candles, orderbook и т.д.)
3	Auth-service	Сервис авторизации на Go, Обрабатывает регистрацию и вход пользователей, выдаёт токены доступа. Взаимодействует с: Frontend, AuthDB
4	AuthDB	База данных пользователей. Хранит логины, хеши паролей, электронную почту и рефреш-токены
5	Kafka	Брокер сообщений. Очередь сообщений между Byb1t-provider и Clickhouse
6	Clickhouse	Служит хранилищем для аналитических данных, хранит и обрабатывает собранные данные с биржи
7	Analytic-service	Сервис, который выполняет аналитические SQL запросы к Clickhouse, и обрабатывает полученные данные

БД была развернута с помощью механизма миграции. Брокер сообщений был развернут вручную через KafkaUI.

Для миграции PostgreSQL и Clickhouse была использована библиотека goose.

Результат развертывания БД и брокера представлен на рисунках 27, 28, 29.

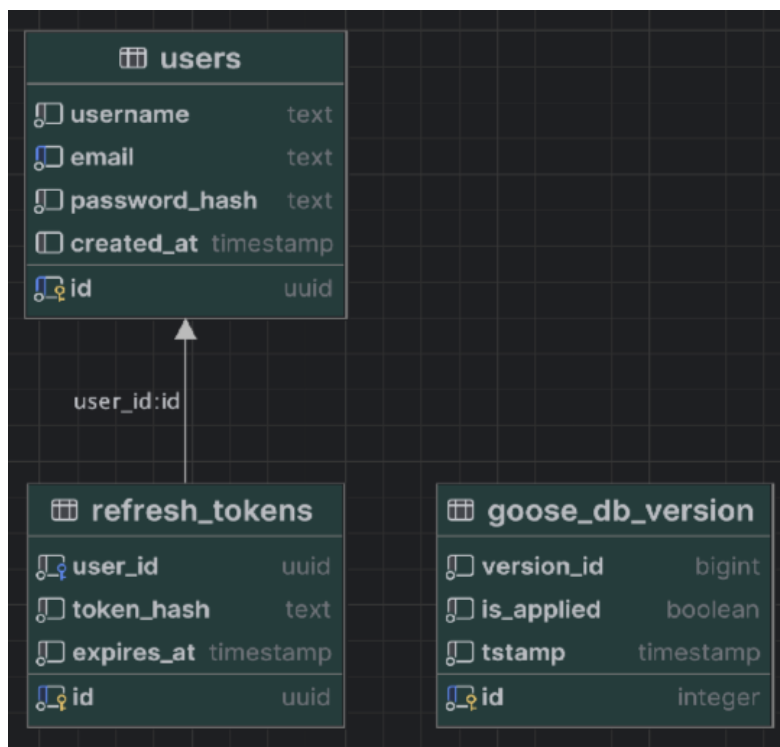


Рисунок 27 – Результат развертывания БД авторизации

На рисунке 27 представлен результат развертывания базы данных авторизации PostgreSQL с использованием библиотеки goose. База данных содержит три компонента. Компоненты базы users, refresh_tokens поддерживают связь один ко многим. А компонент goose_db_version содержит данные о миграциях.



Рисунок 28 – Результат развертывания БД Clickhouse

На рисунке представлен результат развертывания базы данных Clickhouse для аналитики. Содержит материальные представления для сбора данных из Kafka в Clickhouse.

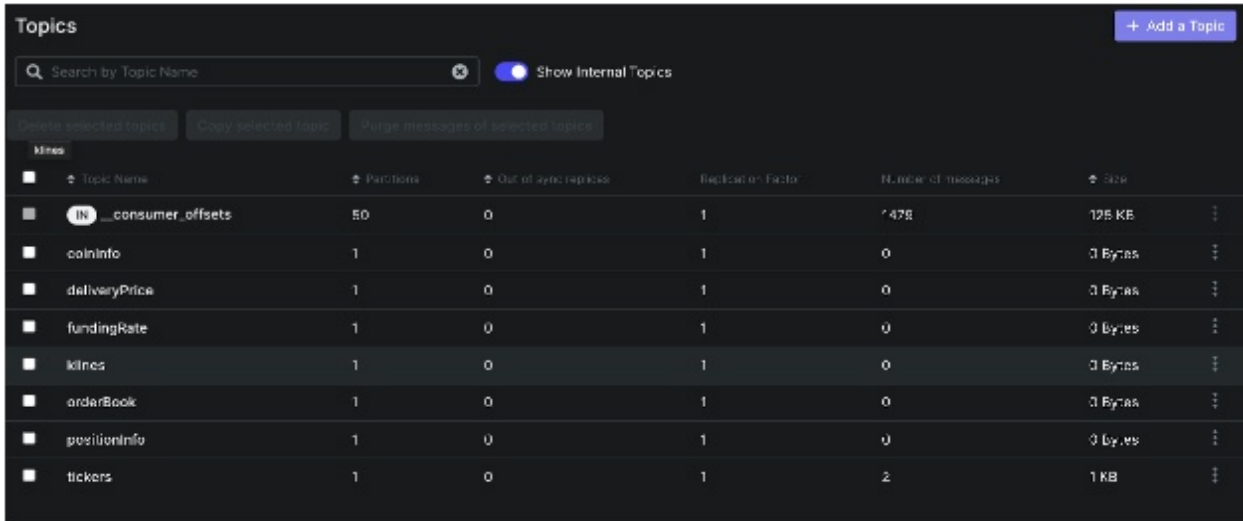


Рисунок 29 – Результат создания топигов Kafka

На рисунке представлен результат создания топигов в брокере сообщений Kafka, из которых читает данные база аналитики Clickhouse.

Далее на рисунке 30 представлена диаграмма развертывания приложения, которая изображает основные компоненты системы

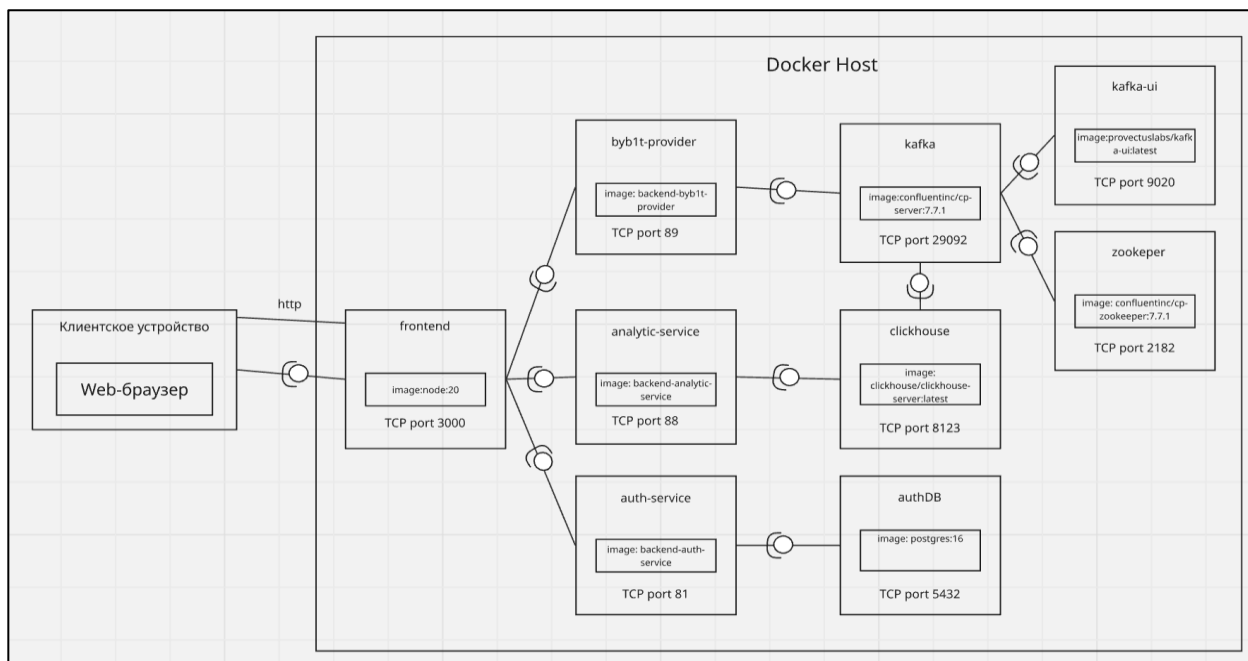


Рисунок 30 – Диаграмма развертывания

На диаграмме видно, что приложение соответствует трех уровневой архитектуре. Сервер приложений и сервер БД обернуты в docker контейнеры, что позволяет в дальнейшем беспрепятственно переносить данное приложение на другие платформы.

Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему, а также предоставляет среду по управлению контейнерами.

Клиентское приложение работает на отдельном сервере, но при этом можно скомпилировать данное приложение и избавиться от данного компонента системы, соединив его с сервером приложений.

4. Контроль качества программного обеспечения

4.1 Функциональное тестирование

Существует достаточно много видов тестирования программного продукта. Были рассмотрены следующие виды тестирования:

- функциональное тестирование;
- модульное тестирование.

Функциональное тестирование – направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом. Данный метод тестирования является одним из самых применяемых, так как позволяет наиболее полно проверить программу на соответствие требованиям.

Модульное тестирование – вид тестов, которые работают на очень низком уровне, близко к исходному коду приложения. Они заключаются в тестировании отдельных методов и функций классов, компонентов или модулей, используемых в ПО. Модульные тесты, как правило, не требуют больших расходов на автоматизацию и могут выполняться сервером непрерывной интеграции очень быстро.

Было принято решение осуществить проверку программы с помощью функционального тестирования и модульного тестирования. Такой выбор обусловлен тем, что к программе были предъявлены в основном функциональные требования, выполнение которых нужно проверить. Ниже приведены листинги выполнения функциональных тестов, представленных в таблице 12.

Таблица 12 – Функциональные тесты RESTfull API приложения

Назначение теста	Описание теста	Результат теста
Тест предназначен для проверки варианта использования «Регистрация»	Входные данные: Почта - test1@mail.ru Логин – test1 Пароль – test1111 Выходные данные: Айди пользователя, добавленного в базу	Тест пройден успешно. Пользователь с новым Id создан в базе пользователей
Тест предназначен для проверки варианта использования «Войти»	Входные данные: Логин – test1 Пароль – test1111 Выходные данные: Токен авторизации	Тест пройден успешно. Пользователю присвоен токен для аутентификации внутри системы

Продолжение таблицы 12

Назначение теста	Описание теста	Результат теста
Тест предназначен для проверки варианта «Запросить данные у поставщика»	Входные данные: Эндпоинт: Tickers Параметры запроса: category=linear&symbol=BT CUSDT Выходные данные: Ответ от поставщика по запросу пользователя в формате JSON	Тест пройден успешно. Пользователь получил данные от поставщика по своему запросу в формате JSON
Тест предназначен для проверки варианта «Провести анализ средней амплитуды свечей»	Входные данные: Дата: 18.03.2025 Выходные данные: Результат проведения анализа в формате таблицы	Тест пройден успешно. Пользователь получил результат анализа в формате таблицы

Листинг 1 – Лог запроса на регистрацию

Request URL: http://localhost: 81/api/register/
 Request Method: POST
 Status Code: 200 OK
 Remote Address: [::1]:81

Body: {
 "email": "test1@mail.ru",
 "username": "test1",
 "password": "test1111"
 }

Response
 {
 "id": "8fb5c16f-9482-4b31-ade7-c3d65e379f00"
 }

На листинге 1 видно, что пользователь смог успешно зарегистрироваться и получить ответ с id пользователя

Листинг 2 – Лог запроса на вход

Request URL: http://localhost:81/api/login/
 Request Method: POST
 Status Code: 200 OK
 Remote Address: [::1]:81

Body: {
 "username": "test1",
 "password": "test1111"
 }

Response
 {
 "access_token":
 "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJyZWZyZXNoX3Rva2VuX2lkIjoimGRkZTJkMDg"

```
tMDcyNC00NTE4LThhYzYtODZkMjEzYjYlZDZiIiwiidXNlcl9pZCI6IjhmYjVjMTZmLTk0ODItNGIz
MSlhZGU3LWMzZDYlZTM3OWYwMCIsInVzZXJfaXAiOiIxNzIuMjEuMC4xIiwiZXhwIjoxNzQ3ODU2N
TgzLzJpYXQiOjE3NDc4NTU2ODN9.Lfy1YbjmUCGPB0nYWwRxrHhkLVUZkQX_Y6Fv0V3Onh2gJe5uo
baEleQBUDL4pXhWro6oQRbYJ3CBTmiYqrU1LA",
  "refresh_token": "M2IWq00XOM/2M0uSm4qLQ5GxxmRkloBlHGvHji7SkQY="
}
```

На листинге 2 видно, что пользователь смог успешно авторизоваться и получить токены для аутентификации.

Листинг 3 – Лог запроса на получение данных от поставщика

```
Request URL:
http://localhost:89/market/tickers?category=linear&symbol=BTCUSDT
Request Method: GET
Status Code: 200 OK
Remote Address: [::1]:89

Response: {
  "ask1Price": "105975.00",
  "ask1Size": "0.017",
  "basis": "",
  "basisRate": "",
  "bid1Price": "105800.00",
  "bid1Size": "0.001",
  "curPreListingPhase": "",
  "deliveryFeeRate": "",
  "deliveryTime": "0",
  "fundingRate": "0.00133471",
  "highPrice24h": "113459.50",
  "indexPrice": "107430.00",
  "lastPrice": "105975.00",
  "lowPrice24h": "104561.00",
  "markPrice": "106347.67",
  "nextFundingTime": "1747872000000",
  "openInterest": "825194.068",
  "openInterestValue": "87757466429.62",
  "preOpenPrice": "",
  "preQty": "",
  "predictedDeliveryPrice": "",
  "prevPrice1h": "106500.00",
  "prevPrice24h": "106930.00",
  "price24hPcnt": "-0.008931",
  "symbol": "BTCUSDT",
  "turnover24h": "28787288.1530",
  "volume24h": "266.1210"
}
```

На листинге 3 видно, что пользователь смог успешно отправить запрос на получение данных от поставщика, и получил ответ в формате JSON.

Листинг 4 – Лог запроса на проведения анализа средней амплитуды свечей

```
Request URL: http://localhost:88/analytics/avg-amplitude?date=2025-03-18
Request Method: GET
Status Code: 200 OK
Remote Address: [::1]:88
```

```

Response: [{symbol: "BTCUSD", hour: "2025-03-18T00:00:00Z", avg_amplitude:
5662.6499999999994, candle_count: 1},...]
[0 ... 99]
[100 ... 199]
    [200 ... 299]
    [300 ... 399]
    [400 ... 499]
    [500 ... 599]
    [600 ... 699]
700:{symbol: "ETHUSD", hour: "2025-04-08T10:00:00Z", avg_amplitude:
3.8599999999999999, candle_count: 1}

```

На листинге 4 видно, что пользователь смог успешно отправить запрос на проведение анализа средней амплитуды свечей и получить ответ от сервера.

4.2 Модульное тестирование

Модульное тестирование позволяет проверить в автоматическом режиме корректность работы отдельных модулей программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Также при модульном тестировании необходимо знать метрику покрытия кода. Она показывает процент исходного кода программы, который был выполнен в процессе тестирования.

Для проведения модульного тестирования была выбрана библиотека `pytest`. Плюсом данной библиотеки является то, что в ней присутствует механизм создания соответствующего окружения для проведения тестирования, что облегчает, например, работу с БД.

Было проведено модульное тестирование подсистемы авторизации и регистрации (таблица 13).

Таблица 13 – Описание модульных тестов

Тестовый метод	Описание	Ожидаемый результат
TestToAvgAmplitudeResponse	Проверяется корректность преобразования структуры AvgAmplitudeDB в структуру ответа AvgAmplitudeResponse. Входные данные: { «Symbol» «BTCUSDT», "Hour": "2025-05-21T12:00:00Z", "AvgAmplitude": 10.5, "CandleCount": 12 }	Ожидаемый результат: Структура AvgAmplitudeResponse с полями: { «symbol»: «BTCUSDT», «hour»: «2025-05-21T12:00:00Z», «avg_amplitude»: 10.5, «candle_count»: 12 }
TestToVolatilityResponse	Проверяется корректность преобразования структуры VolatilityDB в структуру ответа VolatilityResponse. Входные данные: { "Symbol": "ETHUSDT", "AvgVolatility": 1.2, "MaxVolatility": 2.3, "MinVolatility": 0.9, "CandleCount": 30 }	Ожидаемый результат: Структура VolatilityResponse с полями: { "symbol": "ETHUSDT", "avg_volatility": 1.2, "max_volatility": 2.3, "min_volatility": 0.9, "candle_count": 30 }
TestToBullishBearishRatioResponse	Проверяется корректность преобразования структуры BullishBearishRatioDB в структуру ответа AvgAmplitudeResponse. Входные данные: { "Symbol": "BNBUSDT", "Hour": "2025-05-21T13:00:00Z", "BullishCandles": 10, "BearishCandles": 5, "TotalCandles": 15, "BullishRatio": 0.66 }	Ожидаемый результат: Структура BullishBearishRatioResponse с полями: { "symbol": "BNBUSDT", "hour": "2025-05-21T13:00:00Z", "bullish_candles": 10, "bearish_candles": 5, "total_candles": 15, "bullish_ratio": 0.66 }
TestToPriceExtremesResponse	Проверяется корректность преобразования структуры PriceExtremesDB в структуру ответа PriceExtremesResponse. Входные данные: { "Symbol": "SOLUSDT", "MaxPrice7d": 150.0, "MinPrice7d": 120.0 }	Ожидаемый результат: Структура PriceExtremesResponse с полями: { "symbol": "SOLUSDT", "max_price_7d": 150.0, "min_price_7d": 120.0 }

Продолжение таблицы 13

TestToPriceChangePercentResponse	Проверяется корректность преобразования структуры PriceChangePercentDB в структуру ответа PriceChangePercentResponse. Входные данные: { "Symbol": "XRPUSDT", "KlineTimestamp": 16800000000, "Close": 0.5, "PriceChangePercent": 1.23 }	Ожидаемый результат: Структура PriceChangePercentResponse с полями: { "symbol": "XRPUSDT", "kline_timestamp": 16800000000, "close": 0.5, "price_change_percent": 1.23 }
----------------------------------	--	---

Результат прохождения модульных тестов представлен на Рисунке 31

```

✓ Tests passed: 5 of 5 tests – 594 ms
> <4 go setup calls>
=== RUN    TestToAvgAmplitudeResponse
--- PASS: TestToAvgAmplitudeResponse (0.00s)
=== RUN    TestToVolatilityResponse
--- PASS: TestToVolatilityResponse (0.00s)
=== RUN    TestToBullishBearishRatioResponse
--- PASS: TestToBullishBearishRatioResponse (0.00s)
=== RUN    TestToPriceExtremesResponse
--- PASS: TestToPriceExtremesResponse (0.00s)
=== RUN    TestToPriceChangePercentResponse
--- PASS: TestToPriceChangePercentResponse (0.00s)
PASS

```

Рисунок 31 – Результат исполнения модульного тестирования

4.3 Метрики кода

В качестве метрик были подсчитаны метрики устойчивости и абстрактности. Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент.

Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{Ce}{Ca + Ce},$$

где I – мера неустойчивости, которая изменяется в диапазоне от $[0, 1]$ (если $I = 0$, то компонент максимально устойчив, если $I = 1$, то компонент максимально неустойчив);

Ce – количество классов вне данного компонента, зависящих от классов внутри компонента;

Ca – количество классов внутри данного компонента, зависящих от классов вне компонента.

В созданном приложении можно выделить 3 логических компонент:

- Компонент byblt-provider;
- компонент auth-service;
- компонент analytic-service;

Для каждой компоненты была просчитана величина I – метрика неустойчивости.

- $I_1 = \frac{0}{1+0} = 0;$
- $I_2 = \frac{1}{0+1} = 1;$
- $I_3 = \frac{0}{1+0} = 0;$

Метрика A , являющаяся мерой абстрактности компонента, равна отношению количества абстрактных классов или интерфейсов к общему количеству классов в компоненте.

Абстрактностью A называется величина

$$A = \frac{Na}{Nc},$$

где Nc – количество классов в компоненте;

Na – количество абстрактных классов или интерфейсов в нем же.

Метрика A изменяется от 0 до 1, причем 0 означает, что в компоненте вообще нет абстрактных классов или интерфейсов, а 1 – что компонент содержит только абстрактные классы или интерфейсы.

- $A_1 = \frac{4}{16} = 0.25;$

$$\begin{aligned}
 - \quad A_2 &= \frac{5}{10} = 0.5; \\
 - \quad A_3 &= \frac{3}{20} = 0.15;
 \end{aligned}$$

Вычислив меру устойчивости компоненты и ее абстрактность можно отобразить их на графике. На вертикальной оси необходимо отложить меру абстрактности A , а на горизонтальной оси – меру неустойчивости I .

Компонент, который располагаются в верхнем левом углу или в нижнем правом являются максимально качественным. На практике часто случается, так что полученная точка лежит на диагонали между точкой $(1,0)$ и точкой $(0,1)$. Такая диагональ называется главная последовательность (рисунок 4).

Для вычисления количественной оценки используется формула вычисления расстояния до главной последовательности:

$$D = \left| \frac{(A + I - 1)}{\sqrt{2}} \right|,$$

где A – мера абстрактности;

I – мера неустойчивости.

Чем ближе точка располагается к главной последовательности, тем лучше. На рисунке 32 главная последовательность и зоны неприятностей, где компонент является слишком неустойчивым, но при это не является абстрактным и наоборот.

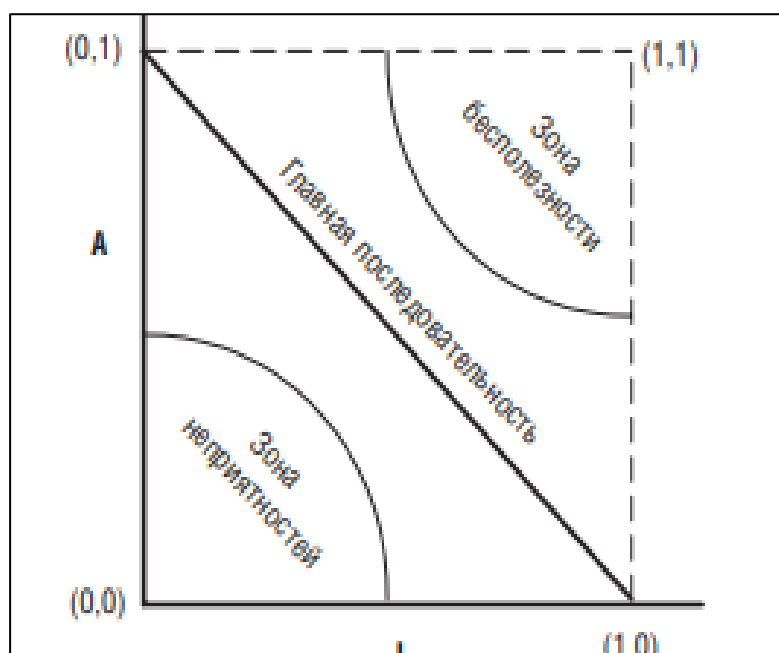


Рисунок 32 – Зоны исключения

Был построен соответствующий график для логических компонент приложения (рисунок 33).

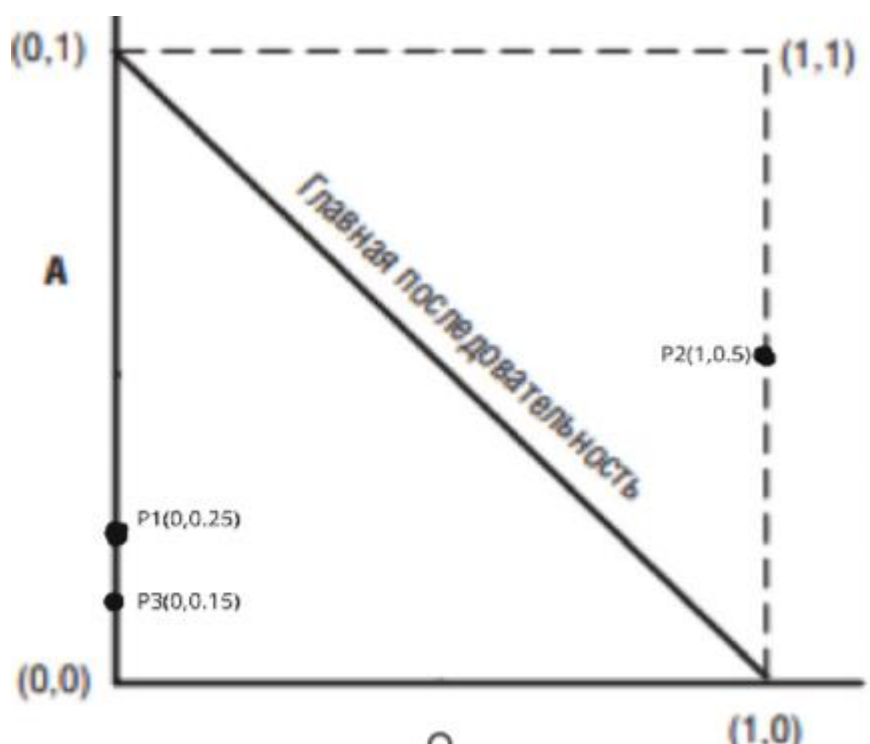


Рисунок 33 – Отображение метрик на графике

Было вычислено расстояние до главной последовательности:

$$- D_1 = \left| \frac{0.25+0-1}{\sqrt{2}} \right| = 0.530;$$

– $D_2 = \left| \frac{0.5+1-1}{\sqrt{2}} \right| = 0.353;$

– $D_3 = \left| \frac{0.15+0-1}{\sqrt{2}} \right| = 0.6;$

Первая и третья компоненты достаточно удалены от главной последовательности и находятся в зоне неприятностей, что связано с тем, что они достаточно изолированы от других компонентов, поэтому это компоненты имеют устойчивость.

Вторая компонента лежит достаточно близко к главной последовательности. Это значит, что отношение абстрактности к устойчивости этой компоненты является правильным.

4.4 Планирование разработки и оценка бюджета

Проведено планирование этапов разработки системы, в результате чего сформирован календарный план, включающий основные задачи, подзадачи и сроки их выполнения.

Ход планирования представлен на рисунке 34 в виде диаграммы Ганта, где визуализированы этапы разработки, их длительность и последовательность выполнения.

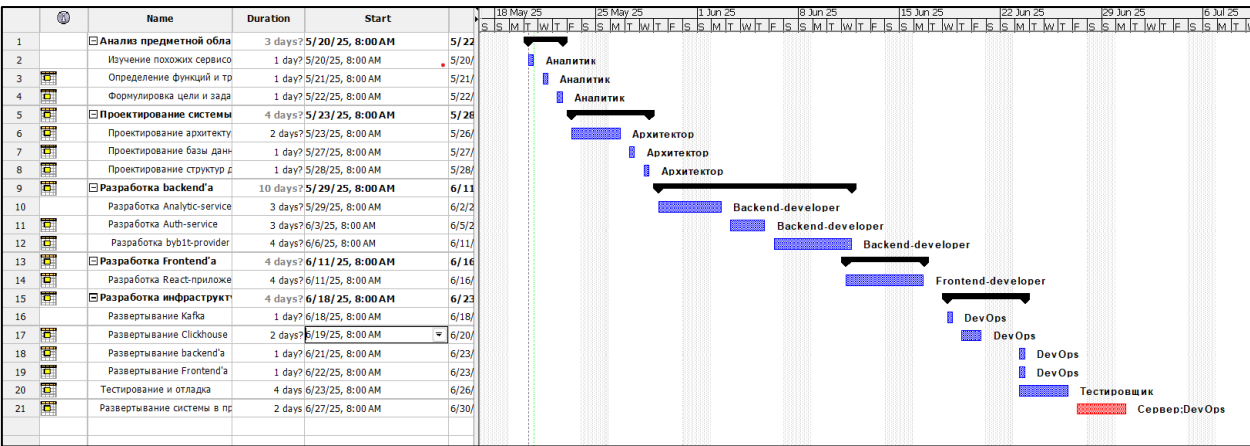


Рисунок 34 – диаграмма Ганта

В таблице представлен расчет бюджета разработки программной системы, и ресурсы, затраченные на ее разработку в продуктовой среде.



Рисунок 35 – Затраты на анализ предметной области



Рисунок 36 – Затраты на проектирование системы

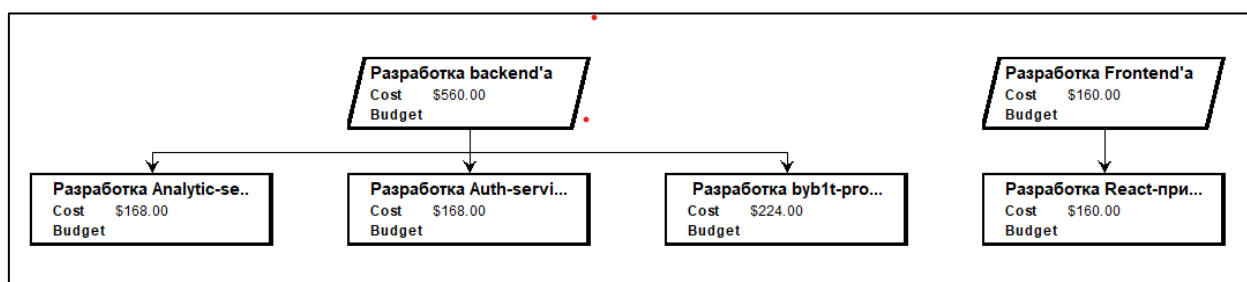


Рисунок 37 – Затраты на разработку backend'а и frontend'а

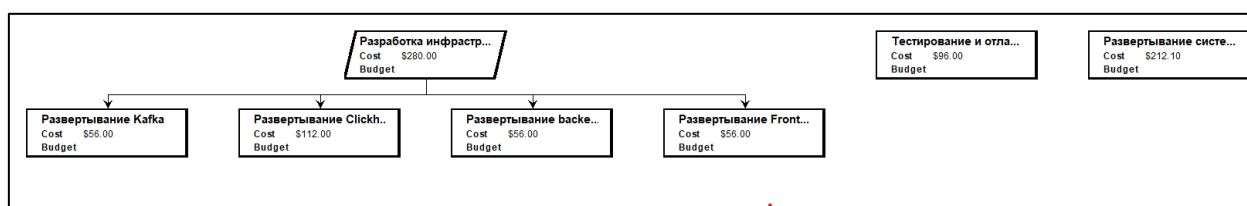


Рисунок 38 – Затраты на развертывание и тестирование системы

В результате расчета бюджета было выявлено, что на реализацию системы придется потратить сумму равную 129480 рублей, и затратить на разработку 25 рабочих дней.

Заключение

В результате выполнения выпускной квалификационной работы была разработана система для анализа и сбора данных крипторынка.

Проведено сравнение аналогов по нескольким критериям, вследствие чего был выбран прототип. Также были описаны аналитические запросы, которыми мы будем обращаться к системе.

На этой основе были выделены функциональные и нефункциональные требования к системе.

Для реализации и решения задачи, поставленной в ходе работы были выбраны архитектура и необходимые технологии. Были спроектированы логическая и физическая схемы БД, а также созданы топики в брокере сообщений. Для основных компонентов системы построены диаграммы классов. Средой разработки для реализации программных средств являются VS Code, где были использованы такие языки программирования как Golang и JavaScript. Разработанное приложение функционирует в любом современном веб-браузере, серверная же часть приложения имеет очень простой способ развертывания, в следствие использования контейнеризации.

Программное обеспечение было протестировано. Для проведения тестирования были выбраны функциональное тестирование и модульное тестирование.

Было проведено планирование и оценка бюджета при помощи программы ProjectLibre. Затраты на проект составили 129 480 рублей. Время разработки 25 рабочих дней.

В результате выполнения выпускной квалификационной работы была разработана система сбора и анализа данных крипторынка, отвечающая всем требованиям технического задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Apache Kafka Use Cases // Apache Kafka [Электронный ресурс]. – URL: <https://kafka.apache.org/uses>
2. Apache Kafka: Benefits and Use Cases // Confluent [Электронный ресурс]. – URL: <https://www.confluent.io/learn/apache-kafka-benefits-and-use-cases/>
3. Building a Real-Time Financial Analytics Platform with ClickHouse // YouTube [Электронный ресурс]. – URL: https://www.youtube.com/watch?v=KrHuGXsS_Gg
4. Building a product analytics solution with ClickHouse // ClickHouse [Электронный ресурс]. – URL: <https://clickhouse.com/blog/building-product-analytics-with-clickhouse>
5. Building A Containerized Microservice in Golang: A Step-by-step Guide with Example Use Case // Velotio [Электронный ресурс]. – URL: <https://www.velotio.com/engineering-blog/build-a-containerized-microservice-in-golang>
6. ClickHouse: to use or not to use? // Medium [Электронный ресурс]. – URL: <https://medium.com/@andriy.lytvynskyy/clickhouse-to-use-or-not-to-use-57965e107815>
7. Complete Guide on Front End Web Development with React // Simpalm [Электронный ресурс]. – URL: <https://www.simpalm.com/blog/front-end-development-using-react>
8. Developing Front-End Apps with React // Coursera [Электронный ресурс]. – URL: <https://www.coursera.org/learn/developing-frontend-apps-with-react>
9. Nison S. Japanese Candlestick Charting Techniques // Penguin Books [Электронный ресурс]. – URL: <https://www.penguinrandomhouse.com/books/120365/japanese-candlestick-charting-techniques-by-steve-nison/> – New York, 1991.

10. Frontend Development using React // Coursera [Электронный ресурс]. – URL: <https://www.coursera.org/learn/frontend-development-using-react>
11. Go and Microservices // Reddit [Электронный ресурс]. – URL: https://www.reddit.com/r/golang/comments/16fi4sb/go_and_microservices/
12. How to Create Microservices — A Practical Guide Using Go // Medium [Электронный ресурс]. – URL: <https://medium.com/@janishar.ali/how-to-create-microservices-a-practical-guide-using-go-35445a821513>
13. PostgreSQL - concepts, benefits and use cases // Tessell [Электронный ресурс]. – URL: <https://www.tessell.com/blogs/postgresql-concepts-benefits-and-use-cases>
14. PostgreSQL Use Cases: Why Data Engineers Love this Database // Airbyte [Электронный ресурс]. – URL: <https://airbyte.com/data-engineering-resources/postgresql-use-cases>
15. React // React Official Website [Электронный ресурс]. – URL: <https://react.dev/>
16. Real-time Analytics with ClickHouse // ClickHouse [Электронный ресурс]. – URL: <https://clickhouse.com/use-cases/real-time-analytics>
17. Real-world use cases and examples of Kafka implementations // Medium [Электронный ресурс]. – URL: <https://medium.com/@psnavya90/real-world-use-cases-and-examples-of-kafka-implementations-37eade7762c5>
18. The Complete Microservices Course in Go // YouTube [Электронный ресурс]. – URL: <https://www.youtube.com/watch?v=KdnxzgSNLTU>
19. What ClickHouse is used for? Use cases overview // Medium [Электронный ресурс]. – URL: <https://medium.com/doublecloud-insights/what-clickhouse-is-used-for-use-cases-overview-b5564e750b83>
20. What is PostgreSQL? How It Works, Use Cases, and Resources // DataCamp [Электронный ресурс]. – URL: <https://www.datacamp.com/blog/what-is-postgresql-introduction>
21. Why using Apache Kafka in real-time processing // Stack Overflow [Электронный ресурс]. – URL:

<https://stackoverflow.com/questions/43402184/why-using-apache-kafka-in-real-time-processing>

Приложение А

(обязательное)

ЛИСТИНГ ПРИЛОЖЕНИЯ

Листинг А.1 – Файл app.go

```
package app

import (
    ctx "context"
    "data-collector/api"
    "data-collector/config"
    "data-collector/kafka"
    "data-collector/service/history"
    "data-collector/service/market"
    "github.com/GiairoZeppeli/utils/context"
    bybit "github.com/wuhewuhe/bybit.go.api"
    "go.uber.org/zap"
    "log"
    "time"
)

const (
    cacheKey      = "coin:%s"
    ttl           = 10 * time.Minute
)

type App struct {
    ctx          context.MyContext
    server       *api.Server
    settings     config.Settings
    kafkaProducer kafka.Producer
}

func NewApp(ctx ctx.Context, logger *zap.SugaredLogger, settings config.Settings) *App {
    return &App{
        ctx:          context.NewMyContext(ctx, logger),
        settings: settings,
```

```

    }
}

func (a *App) InitMQ() error {
    producer, err := kafka.NewProducer(a.settings.Kafka.Address)
    if err != nil {
        log.Fatalf("Failed to create producer: %v", err)
    }
    a.kafkaProducer = producer

    return nil
}

func (a *App) InitService() {
    a.server = api.NewServer(a.ctx)

    bybitClient := bybit.NewBybitHttpClient(PublicApiKey,
        PrivateApiKey, bybit.WithBaseUrl(bybit.TESTNET))

    historyService := history.NewHistoryService(bybitClient,
        a.kafkaProducer)
    marketService := market.NewMarketService(bybitClient,
        a.kafkaProducer)
    a.server.HandleHistory(a.ctx, historyService)
    a.server.HandleMarket(a.ctx, marketService)
}

func (a *App) Run() error {
    go func() {
        if err := a.server.Run(); err != nil {
            a.ctx.Logger.Errorf("error occurred while running
http server: %s", err.Error())
        }
    }()

    a.ctx.Logger.Info("run server")
}

```

```
        return nil
    }

    func (a *App) Shutdown() error {
        err := a.server.Shutdown(a.ctx)
        if err != nil {
            a.ctx.Logger.Errorf("Failed to disconnect from server
%v", err)
            return err
        }

        a.ctx.Logger.Info("server shut down successfully")
        return nil
    }
```

Листинг А.2 – Файл main.go

```
package main

import (
    "context"
    app "data-collector"
    "data-collector/config"
    "fmt"
    "go.uber.org/zap"
    "os"
    "os/signal"
    "syscall"
)

func main() {
    prdLogger, _ := zap.NewProduction()
    defer prdLogger.Sync()
    logger := prdLogger.Sugar()

    fmt.Println(logger.Level())

    mainCtx := context.Background()
    ctx, cancel := context.WithCancel(mainCtx)
    defer cancel()

    settings, err := config.NewSettings()
    if err != nil {
        logger.Errorf("fail to load settings: %s", err)
    }

    app := app.NewApp(ctx, logger, settings)

    if err := app.InitMQ(); err != nil {
        logger.Errorf("fail to init mq: %s", err)
    }
}
```

```
}

app.InitService()
if err = app.Run(); err != nil {
    logger.Errorf("fail to run: %s", err)
    return
}

sigChan := make(chan os.Signal, 1)
signal.Notify(sigChan, syscall.SIGTERM, syscall.SIGINT)
<-sigChan

if err = app.Shutdown(); err != nil {
    logger.Errorf("fail to shutdown: %s", err)
    return
}
}
```

Листинг А.3 – Файл server.go

```

package api

import (
    "data-collector/api/handler"
    "data-collector/api/middleware"
    "data-collector/service/history"
    "data-collector/service/market"
    "github.com/GiairoZeppeli/utils/context"
    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"
    "net/http"
    "time"
)

const (
    maxHeaderBytes = 1 << 20 // 1 MB
    readTimeout    = 10 * time.Second
    writeTimeout   = 10 * time.Second
)

type Server struct {
    httpServer *http.Server
    router      *mux.Router
}

func NewServer(ctx context.MyContext) *Server {
    router := mux.NewRouter()

    allowedOrigins :=
handlers.AllowedOrigins([]string{"http://localhost:3000"})
    allowedMethods := handlers.AllowedMethods([]string{"GET",
"POST", "PUT", "DELETE", "OPTIONS"})
    allowedHeaders := handlers.AllowedHeaders([]string{"Content-
Type", "Authorization"})

```

```

    corsRouter := handlers.CORS(allowedOrigins, allowedMethods,
allowedHeaders)(router)
    recoveryRouter := middleware.RecoveryMiddleware(ctx,
corsRouter)
    wrappedRouter := middleware.UserIdentity(ctx,
recoveryRouter)

    return &Server{
        httpServer: &http.Server{
            Addr:           ":89",
            MaxHeaderBytes: maxHeaderBytes,
            ReadTimeout:    readTimeout,
            WriteTimeout:   writeTimeout,
            Handler:        wrappedRouter,
        },
        router: router,
    }
}

func (s *Server) Run() error {
    return s.httpServer.ListenAndServe()
}

func (s *Server) Shutdown(ctx context.MyContext) error {
    return s.httpServer.Shutdown(ctx.Ctx)
}

func (s *Server) HandlePing(ctx context.MyContext) {
    s.router.HandleFunc("/ping/",
handler.Ping(ctx)).Methods(http.MethodGet)
}

func (s *Server) HandleMarket(ctx context.MyContext, service
market.Service) {

```

```

        s.router.HandleFunc("/market/position/info",
handler.PositionInfo(ctx, service)).Methods(http.MethodGet)
        s.router.HandleFunc("/market/orderbook",
handler.OrderBook(ctx, service)).Methods(http.MethodGet)
        s.router.HandleFunc("/market/funding-rate",
handler.FundingRate(ctx, service)).Methods(http.MethodGet)
        s.router.HandleFunc("/market/delivery-price",
handler.DeliveryPrice(ctx, service)).Methods(http.MethodGet)
        s.router.HandleFunc("/market/coin-info",
handler.CoinInfo(ctx, service)).Methods(http.MethodGet)
        s.router.HandleFunc("/market/tickers", handler.Tickers(ctx,
service)).Methods(http.MethodGet)

    }

func (s *Server) HandleHistory(ctx context.MyContext, service
history.Service) {
    s.router.HandleFunc("/history/candle",
handler.CandleHistory(ctx, service)).Methods(http.MethodGet)
}

```

Листинг А.4 – Файл Service.go

```

package market

import (
    "data-collector/kafka"
    "encoding/json"
    "fmt"
    "github.com/GiairoZeppeli/utils/context"
    bybit "github.com/wuhewuhe/bybit.go.api"
)

const (
    positionInfoTopic = "positionInfo"
    orderBookTopic    = "orderBook"
    fundingRateTopic  = "fundingRate"
    deliveryPriceTopic = "deliveryPrice"
    coinInfoTopic      = "coinInfo"
    tickersTopic       = "tickers"
)

type Service interface {
    GetPositionInfo(ctx context.MyContext, positionInfoParams
map[string]interface{}) (string, error)
    GetOrderBook(ctx context.MyContext, params
map[string]interface{}) (string, error)
    GetFundingRate(ctx context.MyContext, params
map[string]interface{}) (string, error)
    GetDeliveryPrice(ctx context.MyContext, params
map[string]interface{}) (string, error)
    GetCoinInfo(ctx context.MyContext, params
map[string]interface{}) (string, error)
    GetTickers(ctx context.MyContext, params
map[string]interface{}) (string, error)
}

```

```

type Impl struct {
    client    *bybit.Client
    producer  kafka.Producer
}

func      NewMarketService(client    *bybit.Client,      producer
kafka.Producer) *Impl {
    return &Impl{
        client:    client,
        producer:  producer,
    }
}

func      (s    *Impl)      GetPositionInfo(ctx    context.MyContext,
positionInfoParams map[string]interface{}) (string, error) {
    serverResult,      err      :=
s.client.NewUtaBybitServiceWithParams(positionInfoParams).GetPos
itionList(ctx.Ctx)
    if err != nil {
        ctx.Logger.Errorf("GetPositionInfo bybit api err:%v",
err)
        return "", err
    }

    positionInfoResponse := bybit.PrettyPrint(serverResult)
    err      =      s.producer.Produce(positionInfoResponse,
positionInfoTopic)
    if err != nil {
        ctx.Logger.Errorf("GetPositionInfo produce err:%v",
err)
    }

    return positionInfoResponse, nil
}

```

```

func (s *Impl) GetOrderBook(ctx context.Context, params
map[string]interface{}) (string, error) {
    serverResult, err :=
s.client.NewUtaBybitServiceWithParams(params).GetOrderBookInfo(c
tx.Ctx)
    if err != nil {
        ctx.Logger.Errorf("GetOrderbook bybit api err: %v", err)
        return "", err
    }

    resultJSON := bybit.PrettyPrint(serverResult.Result)

    var result OrderbookResult
    if err := json.Unmarshal([]byte(resultJSON), &result); err !=
nil {
        ctx.Logger.Errorf("Failed to unmarshal OrderbookResult
JSON: %v, JSON: %s", err, resultJSON)
        return "", err
    }

    if len(result.Asks) == 0 || len(result.Bids) == 0 {
        ctx.Logger.Errorf("No orderbook data found in result:
asks=%v, bids=%v", result.Asks, result.Bids)
        return "", fmt.Errorf("no orderbook data found")
    }

    ask := result.Asks[0]
    bid := result.Bids[0]

    orderbook := Orderbook{
        Symbol:          result.Symbol,
        Ask1Price:        ask[0],
        Ask1Size:         ask[1],
        Bid1Price:        bid[0],
        Bid1Size:         bid[1],
        CreateTimestamp:  fmt.Sprintf(result.CreateTimestamp),

```

```

        Sequence:          fmt.Sprintf(result.Sequence),
        OrderbookTimestamp: fmt.Sprintf(result.Timestamp),
        UpdateSequence:    fmt.Sprintf(result.UpdateSequence),
        ResponseTime:      fmt.Sprintf(serverResult.Time),
    }

    jsonBytes, err := json.Marshal(orderbook)
    if err != nil {
        ctx.Logger.Errorf("Failed to marshal Orderbook to JSON:
%v", err)
        return "", err
    }
    response := string(jsonBytes)

    if len(response) == 0 {
        ctx.Logger.Errorf("Empty      JSON      generated      from
Orderbook")
        return "", fmt.Errorf("empty JSON")
    }

    err = s.producer.Produce(response, orderBookTopic)
    if err != nil {
        ctx.Logger.Errorf("GetOrderbook produce err: %v", err)
        return "", err
    }

    return response, nil
}

func (s *Impl) GetFundingRate(ctx context.MyContext, params
map[string]interface{}) (string, error) {
    serverResult, err :=
s.client.NewUtaBybitServiceWithParams(params).GetFundingRateHist
ory(ctx.Ctx)
    if err != nil {

```

```

        ctx.Logger.Errorf("GetFundingRates bybit api err: %v",
err)

        return "", err
    }

    resultJSON := bybit.PrettyPrint(serverResult.Result)

    var result FundingRateResult
    if err := json.Unmarshal([]byte(resultJSON), &result); err !=
nil {
        ctx.Logger.Errorf("Failed to unmarshal
FundingRateResult JSON: %v, JSON: %s", err, resultJSON)
        return "", err
    }

    if len(result.List) == 0 {
        ctx.Logger.Errorf("No funding rates found in
result.list")
        return "", fmt.Errorf("no funding rates found")
    }

    fundingRate := result.List[0]

    fundingRateFlat := FundingRateFlat{
        Symbol:          fundingRate.Symbol,
        FundingRate:      fundingRate.FundingRate,
        FundingRateTimestamp: fundingRate.FundingRateTimestamp,
        Category:         result.Category,
        ResponseTime:     fmt.Sprintf(serverResult.Time),
    }

    jsonBytes, err := json.Marshal(fundingRateFlat)
    if err != nil {
        ctx.Logger.Errorf("Failed to marshal FundingRate to
JSON: %v", err)
        return "", err
    }

```

```

    }
    response := string(jsonBytes)

    if len(response) == 0 {
        ctx.Logger.Errorf("Empty JSON generated from
FundingRate")
        return "", fmt.Errorf("empty JSON")
    }

    err = s.producer.Produce(response, fundingRateTopic)
    if err != nil {
        ctx.Logger.Errorf("GetFundingRates produce err: %v",
err)
        return "", err
    }

    return response, nil
}

func (s *Impl) GetDeliveryPrice(ctx context.MyContext, params
map[string]interface{}) (string, error) {
    serverResult, err :=
s.client.NewUtaBybitServiceWithParams(params).GetDeliveryPrice(c
tx.Ctx) // Предполагаемый вызов API
    if err != nil {
        ctx.Logger.Errorf("GetOptions bybit api err: %v", err)
        return "", err
    }

    ctx.Logger.Debugf("Raw serverResult before parsing: %+v",
serverResult)

    resultJSON := bybit.PrettyPrint(serverResult.Result)

    var result OptionResult

```

```

    if err := json.Unmarshal([]byte(resultJSON), &result); err !=
nil {
        ctx.Logger.Errorf("Failed to unmarshal OptionResult
JSON: %v, JSON: %s", err, resultJSON)
        return "", err
    }

    if len(result.List) == 0 {
        ctx.Logger.Errorf("No options found in result.list")
        return "", fmt.Errorf("no options found")
    }

    option := result.List[0]

    optionFlat := OptionFlat{
        Symbol:         option.Symbol,
        DeliveryPrice:  option.DeliveryPrice,
        DeliveryTime:   option.DeliveryTime,
        Category:       result.Category,
        ResponseTime:   fmt.Sprintf(serverResult.Time),
    }

    jsonBytes, err := json.Marshal(optionFlat)
    if err != nil {
        ctx.Logger.Errorf("Failed to marshal Option to JSON:
%v", err)
        return "", err
    }
    response := string(jsonBytes)

    if len(response) == 0 {
        ctx.Logger.Errorf("Empty JSON generated from Option")
        return "", fmt.Errorf("empty JSON")
    }

    ctx.Logger.Debugf("Option before Kafka: %+v", optionFlat)

```

```

ctx.Logger.Infof("Raw JSON before Kafka: %s", response)

err = s.producer.Produce(response, deliveryPriceTopic)
if err != nil {
    ctx.Logger.Errorf("GetOptions produce err: %v", err)
    return "", err
}

ctx.Logger.Infof("Sent to Kafka topic 'options': %s",
response)
return response, nil
}

func (s *Impl) GetCoinInfo(ctx context.Context, params
map[string]interface{}) (string, error) {
    serverResult, err :=
s.client.NewUtaBybitServiceWithParams(params).GetCoinInfo(ctx.Ct
x)
    if err != nil {
        ctx.Logger.Errorf("GetCoinInfo error: %v", err)
        return "", err
    }

    response := bybit.PrettyPrint(serverResult)
    if err := s.producer.Produce(response, coinInfoTopic); err !=
nil {
        ctx.Logger.Errorf("Produce coin info error: %v", err)
    }
    return response, nil
}

func (s *Impl) GetTickers(ctx context.Context, params
map[string]interface{}) (string, error) {
    serverResult, err :=
s.client.NewUtaBybitServiceWithParams(params).GetMarketTickers(c
tx.Ctx)

```

```

if err != nil {
    ctx.Logger.Errorf("GetTickers bybit api err: %v", err)
    return "", err
}

resultJSON := bybit.PrettyPrint(serverResult.Result)

var result TickerResult
if err := json.Unmarshal([]byte(resultJSON), &result); err !=
nil {
    ctx.Logger.Errorf("Failed to unmarshal Result JSON: %v,
JSON: %s", err, resultJSON)
    return "", err
}

if len(result.List) == 0 {
    ctx.Logger.Errorf("No tickers found in result.list")
    return "", fmt.Errorf("no tickers found")
}

ticker := result.List[0]

jsonBytes, err := json.Marshal(ticker)
if err != nil {
    ctx.Logger.Errorf("Failed to marshal Ticker to JSON:
%v", err)
    return "", err
}
response := string(jsonBytes)

if len(response) == 0 {
    ctx.Logger.Errorf("Empty JSON generated from Ticker")
    return "", fmt.Errorf("empty JSON")
}

err = s.producer.Produce(response, tickersTopic)

```

```
if err != nil {  
    ctx.Logger.Errorf("GetTickers produce err: %v", err)  
    return "", err  
}  
  
return response, nil  
}
```

Приложение Б
(рекомендуемое)
ЛИСТИНГ ЗАПРОСОВ БД

Листинг Б.1 – Файл avg_amplitude.sql

```
SELECT
    symbol,
    toStartOfHour(toDateTime(klineTimestamp / 1000)) AS hour,
    avg(high - low) AS avg_amplitude,
    count(*) AS candle_count
FROM default.klines
WHERE toDateTime(klineTimestamp / 1000) >= toDateTime(?)
GROUP BY symbol, hour
ORDER BY symbol, hour;
```

Листинг Б.2 – Файл bullish_bearish_ratio.sql

```
SELECT
    symbol,
    toStartOfHour(toDateTime(klineTimestamp / 1000)) AS hour,
    sum(if(close > open, 1, 0)) AS bullish_candles,
    sum(if(close < open, 1, 0)) AS bearish_candles,
    count(*) AS total_candles,
    sum(if(close > open, 1, 0)) / count(*) AS bullish_ratio
FROM default.klines
WHERE toDateTime(klineTimestamp / 1000) >= toDateTime(?)
GROUP BY symbol, hour
ORDER BY symbol, hour;
```

Листинг Б.3 – Файл price_change_percent.sql

```
SELECT
    symbol,
    klineTimestamp,
    close,
    (close - lagInFrame(close, 1) OVER (PARTITION BY symbol ORDER
    BY klineTimestamp)) / lagInFrame(close, 1) OVER (PARTITION BY
    symbol ORDER BY klineTimestamp) * 100 AS price_change_percent
FROM default.klines
WHERE toDateTime(klineTimestamp / 1000) >= toDateTime(?)
```

```
ORDER BY symbol, klineTimestamp;
```

Листинг Б.4 – Файл price_extremes.sql

```
SELECT
    symbol,
    max(high) AS max_price_7d,
    min(low) AS min_price_7d
FROM default.klines
WHERE toDateTime(klineTimestamp / 1000) >= toDateTime(?)
GROUP BY symbol
ORDER BY symbol;
```

Листинг Б.5 – Файл volatility.sql

```
SELECT
    symbol,
    avg(high - low) AS avg_volatility,
    max(high - low) AS max_volatility,
    min(high - low) AS min_volatility,
    count(*) AS candle_count
FROM default.klines
WHERE toDateTime(klineTimestamp / 1000) >= toDateTime(?)
GROUP BY symbol
HAVING avg_volatility IS NOT NULL
ORDER BY symbol, avg_volatility DESC;
```