Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РтФ
Кафедра/департамент информационных технологий и систем управления

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Заведующий кафедрой
_____ Е.В. Кислицын
(подпись) (Ф.И.О.)
«31» 05 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка клиент-серверной части приложения для торговли на бирже Binance»

Научный руководитель: Юманова Ирина Фарисовна
к.ф.-м.н., доцент кафедры ИТиСУ

Нормоконтролер: Абакумова Анна Геннадьевна
подпись

Студент группы: Роман Игоревич Новиков
подпись

Екатеринбург
2024

РЕФЕРАТ

Тема выпускной квалификационной работы: Разработка клиент-серверной части приложения для торговли на бирже Binance.

В состав ВКР входят: текст на 76 с., 30 рис., 4 таблицы, 66 источников, 6 приложений.

Ключевые слова: криптовалюта, трейдинг, биржа, разработка, построение архитектуры, python, django, vue.js, docker, postgresql.

Цель работы – проектирование и разработка клиент-серверного web-приложения, позволяющего осуществлять торговлю на бирже Binance.

Объектом выпускной квалификационной работы является набор торговых действий трейдера и его взаимодействие с криптовалютной биржей в процессе торговли.

Предметом выпускной квалификационной работы является процесс разработки клиент-серверной части web-приложения для торговли на бирже Binance.

В работе рассматривается внутридневной трейдинг, как отдельное ответвление торговли на бирже, проводится сравнение с «Личным кабинетом трейдера» – продуктом от команды ООО «Ватага». Анализируются подходы к построению архитектуры клиент-серверных приложений, актуальный на сегодняшний день технологический стек для разработки web-приложений. Описывается процесс проектирования и разработки, а также рефакторинга и тестирования приложения. Отдельно описываются подходы к обеспечению безопасности и перспективы развития web-приложения для торговли на бирже Binance.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

MVP – Minimal Viable Product – это тестовая версия товара, услуги или сервиса с минимальным набором функций (иногда даже одной), которая несет ценность для конечного потребителя.

DEX – Decentralized Exchange – децентрализованная биржа.

URL – Uniform Resource Locator – путь к сайту/документу в сети интернет.

API – Application Programming Interface – это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными. Может употребляться в словосочетании «API эндпоинт» – это конкретный URL.

HTTP – HyperText Transfer Protocol – это протокол передачи гипертекста, который используется для связи между веб-браузерами и веб-серверами.

UDP – User Datagram Protocol – это один из немногих ключевых элементов набора сетевых протоколов для Интернета.

IIFE – Immediately Invoked Function Expression – это JavaScript функция, которая выполняется сразу же после того, как она была определена.

ORM – Object Role Model – это технология, которая позволяет разработчикам работать с базами данных с использованием объектно-ориентированного подхода.

NDA – Non-Disclosure Agreement – соглашение о неразглашении.

Observer – дословный перевод: «наблюдательный пункт», фактически, это сервис, который позволяет следить за работоспособностью какой-либо системы в прямом эфире.

Брокер сообщений – это приложение, которое преобразует сообщение по одному протоколу от приложения-источника в сообщение протокола приложения-приёмника, тем самым выступая между ними посредником.

ВВЕДЕНИЕ

Актуальность темы исследования. Сегодня многие традиционные формы заработка переходят в информационную, более технологическую, сферу. В связи с развитием информационных и коммуникационных технологий спектр возможностей для дистанционной работы онлайн значительно увеличился и стал широко использоваться в различных сферах деятельности. Торговля на финансовых биржах не стала исключением. Вопреки существенным опасениям, высказанным участниками глобального финансового рынка относительно стабильности криптовалютных активов, последние демонстрируют тенденцию к закреплению своих позиций в качестве ключевого элемента будущих экономических отношений. Как и в других видах торговли, трейдеры по всему миру пытаются предсказать движение того или иного актива и заработать на разнице между покупкой и продажей. С накоплением опыта явно обозначился отдельный вид такой торговли – внутридневной трейдинг, в рамках которого все сделки закрываются вместе с закрытием рынка, без переноса на следующий торговый день [1].

Так как это относительно новое направление, на данный момент существует мало систем, обеспечивающих пользователю доступ к данной сфере деятельности. Специализированного программного обеспечения, предоставляющего единый удобный и, что самое главное, быстрый доступ к инструментам для анализа и упрощения работы (например, путем добавления наглядности) еще меньше. Единственным аналогичным решением на рынке СНГ является web-приложение от компании ООО «Ватага», рассмотренное в разделе 1.2 данной работы.

Актуальность проектирования и реализации сервиса для торговли подтверждается тем, что в наше время по всему миру растет количество трейдеров, инвестирующих в криптоактивы, что подтверждается ростом стоимости криптовалют, обеспеченных фиатной валютой [2].

Гипотеза исследования состоит в том, что при использовании подходящего технологического стека и верной архитектуры, получится не только реализовать web-приложение для торговли на бирже, но и получить более быстрые call-back ответы со стороны биржи при запросе на закрытие открытых позиций, снятие лимитных заявок, обновление данных по текущей цене позиции и отправке данных между спотовым и фьючерскными счетами по сравнению с аналогом – web-приложением от компании ООО «Ватага».

Целью исследования являются проектирование и разработка веб-приложения для высокочастотной торговли криптовалютами с целью повышения скорости и эффективности операций по сравнению с существующими аналогами.

Для достижения поставленной цели необходимо решить следующие **задачи**:

– проанализировать предметную область и существующие аналоги, определиться с методологией разработки продукта;

– на основе анализа существующих подходов и технологий, определиться с технологическим стеком для разрабатываемого сервиса, а также построить модель архитектуры;

– на основании выбранных технологий и модели архитектуры, разработать web-приложение для торговли на бирже Binance, провести рефакторинг и тесты;

– описать подходы к безопасности и перспективы развития разработанного web-приложения.

Объектом процесс торговли криптовалютами на бирже, с акцентом на внутридневной торговле.

Предметом является разработка архитектуры и функционала веб-приложения для внутридневной торговли криптовалютами, ориентированного на ускорение операций и повышение эффективности за счет оптимизации взаимодействия с биржевыми API и использования современных технологий.

Методы исследования включают в себя:

– анализ, сравнение, систематизацию и обобщение данных о существующих приложениях для внутридневного трейдинга, а также технологиях для разработки в целом;

– апробацию современных баз данных, frontend и backend технологий при создании приложения с Web-интерфейсом;

– апробацию разработанного web-приложения на реальных пользователях;

– анализ и обобщение данных, полученных во время апробации продукта.

Теоретической основой исследования стали:

– решения конкурентов;

– современные концепции и технологии разработки бэкенд и фронтенд частей, базы данных, а также организации обменом данными;

– документация к различным используемым в современном программировании фреймворкам и библиотекам;

– документация к различным АПИ эндпоинтам (API endpoints).

– научными статьями.

База исследования представлена данными, полученными с API биржи Binance, обобщенной статистикой торговли трейдеров, контрольными замерами быстродействия аналогичного web-приложения.

Практическая значимость исследования заключается в проведении анализа существующих специализированных под нужды внутридневных трейдеров для торговли на бирже Binance. Также реализовано приложение для торговли на бирже Binance, произведены замеры быстродействия получившейся системы, проведена апробация на реальных пользователях с последующим обобщением, систематизацией и анализом полученных данных.

Структура работы. Во введении обоснована актуальность выбора темы, поставлены цели и задачи исследования, определен предмет и объект

исследования, охарактеризованы методы исследования, описана теоретическая основа.

В первой главе описана и проанализирована предметная область, основные понятия, существующие аналоги, контекст выполнения работ, а также методология разработки и причины ее использования.

Во второй главе описан технологический стек проекта: анализ использованных инструментов разработки, обоснование выбора технологий, в контексте которых представлена архитектура продукта.

В третьей главе представлен процесс разработки, описаны использованные библиотеки и механизм работы web-приложения, а также результаты апробации и тестирования разработанного клиент-серверного приложения.

В заключении сформулированы краткие выводы относительно выбранных подходов к разработке, технологического стека и разработанного приложения в целом.

В работе имеется библиографический список, включающий 68 наименований, 7 приложений.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Введение в предметную область

Внутридневная торговля (она же – дей-трейдинг, intraday) – это краткосрочная торговля внутри дня, при которой сделки открываются и закрываются во время одной сессии. Пойдет цена вниз или вверх – значения не имеет. В конце дня и прибыльные, и убыточные позиции должны быть закрыты [3]. Внутридневной трейдер держит позицию от нескольких секунд до одного-двух часов и зарабатывает на незначительных движениях цены, обычно не превышающих 1%.

Среди внутридневного трейдинга выделяют несколько основных стратегий торговли, например: торговля на новостях, трендовая торговля, скальпирование (или скальпинг). Стратегия скальпирования основана на незначительных изменениях цены акций в течение дня и частых входах в рынок и выходах из него в течение торговой сессии. Под торговой сессией подразумевается период, в который трейдер осуществляет покупки или продажи активов. Данная стратегия пользуется популярностью на рынке криптовалют из-за его высокой волатильности. Скальперы часто используют кредитное плечо [4] для открытия большого количества сделок, а также стоп-лоссы для управления рисками [5].

Для внутридневных трейдеров, занимающихся скальпингом основными критериями при выборе web-сервиса будут являться:

– своевременное и быстрое обновление данных;

– отсутствие нерелевантных данных, возможность зайти или закрыть позицию ежесекундно;

– возможность создавать отложенные заявки, исполняемые при достижении определенной цены для получения выручки и минимизации убытка;

– обеспечение непрерывности торговой сессии;

– возможность одновременной торговли большого количества торговых пар;

– обновление информации с биржи в режиме реального времени, синхронизация с биржей;

– простой и удобный функционал для входа и выхода из сделки;

– безопасность личных данных.

Сейчас возможность торговать криптовалютой предоставляет большое количество бирж, но особый интерес представляют централизованные биржи.

Централизованные биржи (CEX) – это самый часто встречающийся на сегодняшний день способ организации торговли в криптосфере. Управление такими биржами осуществляет как правило лицо или группа лиц, которое контролируют платформу и взимают с клиентов определенную плату, или комиссию, за предоставленную услугу в виде проведения сделки [6]. Из-за большого количества крупных игроков, например: Binance, Coinbase Pro, Kraken, Upbit и прочих, на рынке централизованных бирж сильная конкуренция [7].

Децентрализованные платформы, созданы на технологии распределенного реестра и функционируют при помощи смарт-контрактов. Децентрализованные криптовалютные биржи (DEX) отличаются тем, что на них отсутствует централизованное руководство в лице одного или нескольких человек. Это платформа, децентрализующая базовые функции биржи, включая торговлю/обмен активов, сверку ордеров и размещение депозитов [8].

1.2 Анализ конкурентных решений

Параллельно с развитием информационных технологий происходит общая диджитализация общества, такие виды занятости, как торговля на бирже, переходит на новый этап развития в связи с появлением современных, более удобных направлений в экономическом секторе, например

криптоактивы, а также в связи с появлением новых более удобных сервисов по автоматизации рутинного процесса торговли.

Трейдеры, торгующие на биржах, нацелены на получение максимальной прибыли и автоматизацию процессов торговли. Таким образом, на рынке есть несколько решений, позволяющих увеличить ежедневный доход трейдера за счёт удобства и автоматизации.

Сейчас, по многим причинам, ситуацию с развитием криптовалюты в России можно назвать сложной [9]. Данная финансовая область слабо регламентирована со стороны законов и нормативно-правовых актов. В ходе анализа рынка криптовалют в РФ была найдена только одна компания – ООО «Ватага», которая предоставляет схожий функционал, необходимый для внутридневных трейдеров. Однако весь перечисленный функционал имеется внутри самой биржи Binance, что позволяет работать напрямую с биржей.

Для анализа продуктов, предоставляемых ООО «Ватага» и Binance, а также выявления их сильных и слабых стороны был выбран метод стратегического планирования – SWOT анализ [10]. По итогам проведения анализа была составлена таблица [11], основываясь на данных, которой, можно сделать вывод, что к выбранному сегменту есть интерес со стороны конечных потребителей, но на рынке нет продукта, удовлетворяющего всем запросам, а те, что есть имеют существенные недостатки. Как следствие, было принято решение спроектировать и разработать клиент-серверное web-приложение для торговли на бирже Binance.

1.3 Анализ методологий разработки

Перед стартом любого крупного и амбициозного проекта команде необходимо определиться с тем, как идеологически и фактически вести предстоящую деятельность. На сегодняшний день существует более пяти различных методологий ведения проектов: Waterfall, Agile, Kanban, Lean, PRINCE2 и другие. Под проектом следует понимать временное предприятие,

предназначенное для создания уникальных продуктов, услуг и результатов [12]. Среди методологий встречаются как классические методы, например, Waterfall [13] – это линейный или каскадный подход к ведению проектной деятельности, так и более современные методы, например, Agile-подходы. Некоторые agile-подходы подразумеваю итерации длительностью 1–2 недели с демонстрацией достижений в конце каждой из них. Команда проекта определяет содержание, которого они могут достигнуть, на основе приоритизированного бэклога, оценивает требуемый объем работы и сотрудничает на протяжении итерации над разработкой содержания [14].

В рамках работы были наиболее полно рассмотрены две вышеназванные методологии ведения проектной деятельности: гибкая методология (Agile) и каскадная модель (Waterfall). Результаты анализа указанных методологий представлены в приложении А.

Таким образом, наглядно можно выделить основные преимущества Agile методологии:

– Он позволяет команде разработки быстро реагировать на изменения рынка благодаря итеративному подходу, что, в свою очередь, позволяет нивелировать риск устаревания продукта и делает продукт более конкурентноспособным;

– Благодаря регулярным встречам и демонстрациям, позволяет команде собрать необходимую обратную связь от заказчика и, в случае необходимости, скорректировать курс путем пересмотра приоритетов и функционала;

– Способствует повышению мотивации команда разработки за счёт более автономной работы, возможности принимать самостоятельные решения в моменте и наглядной демонстрации проделанной работы, что помогает поддерживать оптимистичный настрой и уровень удовлетворенности;

С учетом всех преимуществ можно утверждать, Agile – лучшая из доступных методологий ведения проектов, аналогичных разрабатываемому. Кроме того, отметим, что Agile имеет несколько фреймворков, например:

Kanban и Scrum. Kanban подразумеваем наличие одноименной доски со столбцами «сделать», «в процессе», «тестирование», «сделано» (либо другие вариации по усмотрению команды), по которой каждый разработчик в любой момент времени может перемещать карточку с задачей. A Scrum основывает на итеративном подходе, разделяя работа на короткие промежутки – спринты, которые обычно длятся от двух до четырех недель. По итогам каждого из спринтов команда представляет результат своей деятельности. Для работы был выбран Scrum – это фреймворк, предназначенный для решения нетривиальных задач, для эффективного и творческого создания продуктов с максимально возможной ценностью. Выбор обусловлен маленьким количественным составом команды, необходимостью регулярной обратной связи от всех участников и желанием постоянно улучшать процесс разработки, что обеспечивается проведением ретроспективы по итогам спринта. Ретроспектива – это регулярно проводимый семинар, в ходе которого участники анализируют свою работу и полученные результаты с целью улучшения как процесса, так и самого продукта [16].

Что касается применения фреймворка Scrum, было принято решение оставить все ключевые артефакты: бэклог продукта – упорядоченный по приоритету и обновляемый список задач, направленных на разработку и улучшение продукта; бэклог спринта – список заданий из общего бэклога продукта, которые были переданы для выполнения во время ближайшей итерации; инкримент продукта – это готовые к поставке результаты работы над заданиями из прошедшего спринта [17]. Кроме основных артефактов в работу были введены критерии готовности задачи (Definition of Done) [18], что позволило обеспечить единое и четкое понимание готовности, а также определить границы инкремента.

2. ПОСТРОЕНИЕ АРХИТЕКТУРЫ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

2.1 Архитектура приложения

2.1.1 Общая архитектура

В ходе исследования рынка были определены ключевые функции приложения, которые лягут в основу MVP [19]. Как и для многих других онлайн-продуктов на рынке, в первую очередь требовалось реализовать регистрацию, авторизацию и аутентификацию. Кроме того, необходимо было наладить стабильный и быстрой обмен данными непосредственно с биржей Binance, что, в свою очередь, подразумевало взаимодействие с API ключами и websocket’ами. Так как продукт позволяет оперировать денежными средствами, очень важно было обеспечить его отказоустойчивость и надежность к различного рода попыткам взлома.

Разработка архитектуры является одним из самых важных этапов в разработке программного продукта. При проектировке информационных систем определяется структура данных, их организация и способы хранения, с целью обеспечения эффективного и надежного управления информацией [20].

Проектирование и разработка современных web-приложений требует соблюдения определенных принципов, чтобы обеспечить безопасность, масштабируемость, легкость в поддерживании и тестировании. На основании этих критериев, было принято решение придерживаться идеологии, разработанной Робертом Мартином (известным под псевдонимом Uncle Bob) «The Clean Architecture» [21]: разделять бизнес-логику и функциональность, разделение на слои, каждый из которых отвечает за четко определенный аспект приложения, использование исключительно чистых границ между слоями. В совокупности эти принципы позволяют создать крупный проект, который не только легко поддерживать и масштабировать, но и легко

разрабатывать и тестировать. Во-первых, продукт разделен на микросервисы, каждый из которых отвечает за определенную часть функционала. Во-вторых, внутри микросервисов допускается деление на виджеты, которые можно включать или отключать при необходимости. Также, для обеспечения дополнительной надёжности сервиса, было решено реализовать дополнительный сервис, который является наблюдателем за работоспособностью для всех остальных частей проекта.

Абсолютно все части продукта должны быть обёрнуты в Docker [22] контейнеры, что обеспечит независимость окружения внутри каждого отдельного контейнера, но не ограничит взаимодействие между ними. Кроме того, Docker также прост в реализации и удобен вовремя развертывания продукта на серверах.

Программное обеспечение архитектуры «клиент-сервер» состоит из двух частей: программного обеспечения сервера и программного обеспечения пользователя – клиента [23]. Клиентская часть выполняется на стороне пользователя и отвечает за пользовательский интерфейс. Обычно она состоит из HTML, CSS и JS, отвечает за логику продукта, исполняемую на стороне пользователя, а также визуальное представление, с которым взаимодействует пользователь. Серверная часть содержит непосредственно бизнес-логику, выполняет взаимодействие с базами данных и внешними системами, обрабатывает HTTP/HTTPS/WebSocket запросы. О различиях этих видов соединений можно прочитать в пункте 3.1 «Технический стек клиентской части приложения. Схема взаимодействия».

Из-за высоких репутационных и финансовых рисков, существовала потребность в разработке отдельного сервиса, который бы осуществлял мониторинг состояния всех остальных микросервисов, что, в свою очередь, обеспечил бы быстрое информирование с последующим реагирование в случае, если какая-либо часть клиент-серверного приложения либо совсем перестала работать, либо начала выдавать ошибки.

Таким образом, разбив весь продукт на отдельные компоненты, удастся не только добиться поддерживаемости, читаемости и логического разделения на блоки, но и обеспечить надежную, отзывчивую систему, устойчивую к высоким нагрузкам и возможным отказам. С общей архитектурой приложения можно ознакомиться в приложении Б.

Отдельное внимание следует уделить, что абсолютно все сервисы были обернуты в отдельные Docker контейнеры. У этого есть несколько причин:

1) могут возникнуть проблемы на этапе отладки приложения. Повышается сложность в определении сервиса, в котором возникла какая-либо ошибка, а также трудно изолировать проблему, не оказав влияние на другие сервисы внутри контейнера;

2) дополнительные трудности при обновлении. У разных сервисов могут быть разные зависимости от сторонних библиотек. Таким образом, объединив все или часть контейнеров, при обновлении какого-либо сервиса, придется обновлять и другие, при этом, уделять дополнительное внимание потенциальным ошибкам;

3) каждый сервис потребляет разное количество аппаратных ресурсов. Объединение может привести к неэффективному использованию памяти или ресурса процессора;

4) повышенный риск уязвимости web-приложения. В случае, если злоумышленник найдет уязвимость в одном из сервисов, любой другой сервис внутри этого контейнера будет под угрозой;

5) отсутствие гибкости и потенциала масштабирования. Вся архитектура web-приложения рассчитана на потенциальное масштабирование. В случае, если часть сервисов будет находиться в одном контейнере, придется масштабировать весь контейнер целиком, а не конкретный сервис, что не всегда необходимо.

Из этого можно сделать вывод, что хоть Docker и предоставляет возможность развернуть все сервисы в одном контейнере, лучшим решением будет именно разделение по принципу: «один сервис – один контейнер».

2.1.2 Архитектура серверной части приложения

Любая серверная часть web-приложения состоит из:

– веб-сервера – позволяет принимать и обрабатывать различного типа запросы, возвращая, чаще всего, статические файлы, такие как: HTML, CSS, JavaScript;

– бизнес-логики – отвечает за непосредственную логику обработки запросов от клиента. Эта же часть отвечает за реализацию функциональности приложения;

– базы данных – хранилище данных, необходимых для работы приложения;

– API – интерфейс, позволяющий клиентской части взаимодействовать с бизнес-логикой и с backend в целом;

Существуют различные архитектурные паттерны backend, обладающие как преимуществами, так и недостатками. Хотя разновидностей очень много, ниже разобраны наиболее часто встречающиеся из них:

1) Model-View-Controller, как следует из названия, разделяет приложение на три соответствующих основных компонента [24]:
    a. модель (Model) – отвечает за представление данных и бизнес-правила. Модель не зависима от View или Controller,
    b. представление (View) – отвечает за отображение данных. Никакой логики обработки данных не содержит,
    c. контроллер (Controller) – отвечает за обработку запросов пользователя. Фактически, осуществляет координацию между моделью и представлением.

    В качестве преимуществ можно выделить:
    – разделение ответственности,
    – гибкость и расширяемость,

– тестируемость.

К ключевым недостаткам относят:

– сложность разработки,

– избыточность кода.

Со схемой работы MVC можно ознакомиться на рисунке 1.

2) Model-View-ViewModel (MVVM) [25] – является разновидностью MVC, но содержит ряд особенностей:

a. модель (Model) – также отвечает за бизнес-логику и данные, но, в отличии от Model в MVC паттерне, сконцентрирована на хранении и обработке данных без прямого взаимодействия с представлением;

b. представление (View) – также отвечает за отображение данных пользователю,

c. содель представления (ViewModel) – является прослойкой между моделью и представлением. Фактически, выполняет функцию сериализатора, то есть, предоставляет данные из модели в формате, удобном для представления. Кроме того, может обеспечивать определенную логику, связанную с пользовательским интерфейсом.

В качестве преимуществ можно выделить:

– более высокая степень абстракции,

– легкость тестирования,

– обеспечение гибким управлением представлений;

В качестве недостатков:

– еще больше кода для реализации,

– потенциальное усложнение отладки.

Со схемой работы можно ознакомиться на рисунке 2.

3) Model-View-Template (MVT) также является разновидностью MVC с некоторыми особенностями: вместо контроллера (Controller) используется шаблон (Template) [26]. Шаблон отвечает за представление данных пользователю. Фактически, чаще всего это HTML-разметка с применением

вставок template tag, которые позволяют вставить данные из модели и контролировать логику отображения.

Преимущества:

- простота и скорость разработки,
- гибкость

Недостатки:

- сложность в масштабировании,
- ограниченная гибкость,
- трудности с соответствием принципам DRY («Do not Repeat Yourself» – принцип, в соответствии с которым должна быть возможность переиспользования кода, а не дублирования).

4) Event-Driven Architecture (EDA) – это архитектура, основанная на возникающих событиях и реакциях на них [27]. Существует несколько артефактов такого подхода:

   a. событие (Event) – своеобразный сигнал о наступлении целевого изменения в системе. Могут посылаться как изнутри, так и снаружи (например, от пользователя);

   b. подписка (Subscription) – различные компоненты могут подписываться на определенные события или типы событий. Подписки могут быть как одноразовыми, так и постоянными;

   c. публикацию (Publish) – компоненты имеют возможность публикации событий.

   Преимущества:

   - асинхронность и скорость реакции,
   - гибкость,
   - независимость компонентов.

   Недостатки:

   - потеря согласованности из-за асинхронности,
   - сложность разработки и тестирования,

– сложность в текущем мониторинге

На рисунке 3 представлен пример архитектуры приложения с применением EDA паттерна. На представленном рисунке описывается покупка дополнительного функционала на сайте. В одном микросервисе осуществляется непосредственно оплата, затем сообщение о событии попадает в брокер сообщений, откуда уходит в два других микросервиса: отправка уведомления или чека на почту и изменение интерфейса для конкретного пользователя (например, открыть доступ на ранее недоступную страницу).

Было принято решение остановиться на MVC, так как у команды уже был опыт в построении архитектур согласно этому паттерну.

Важно отметить, что изначально было принято решение не включать сервис мониторинга (observer) в эту систему и не располагать его в Docker [28] контейнере в рамках поднятого приложения. Объясняется это тем, что необходимо было иметь абсолютно абстрагированную систему, которая сможет наблюдать за состоянием контейнеров с сервисами, но, в случае его, не откажет сама. Именно из этих побуждений систему мониторинга доступности необходимо было развернуть на стороннем сервере как отдельный продукт. Кроме того отмечу, что на рынке существует большое количество уже разработанных систем мониторинга: AppOptics Docker Monitoring [29], Promtheus [30] и другие. Все они имеют GUI и визуализацию данных, но в рамках проекта эти решения выглядели избыточными и слишком ресурсозатратными. Кроме того, часть из них не предоставляют открытой лицензии на использование, что делает их использование невозможным без регулярной или единоразовой оплаты, что влечет за собой дополнительные издержки. Исходя из этого, было принято решение использовать самописную систему мониторинга, которая была разработана в рамках стороннего продукта.

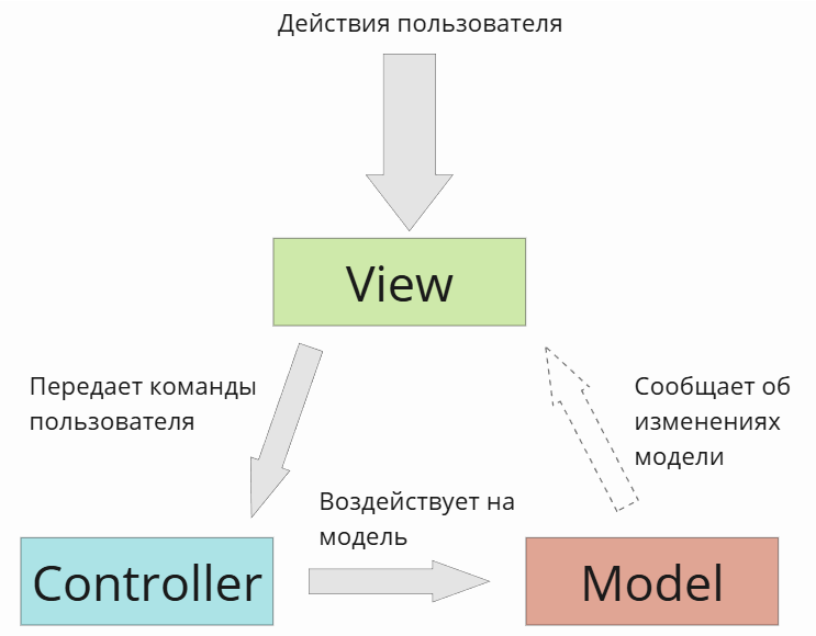


Рисунок1 – Схема работы паттерна проектирования MVC.

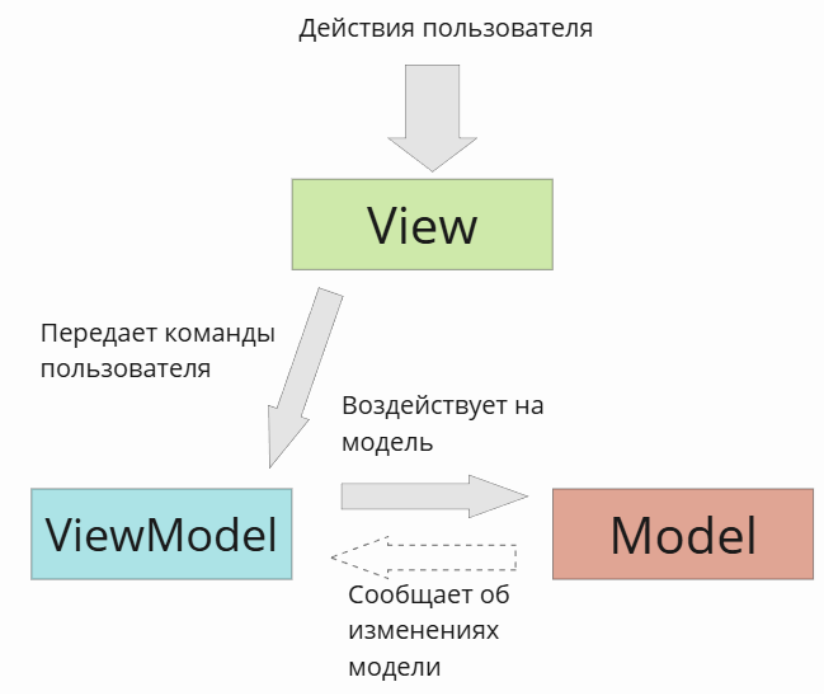


Рисунок 2 – Схема работы паттерна проектирования MVVM.

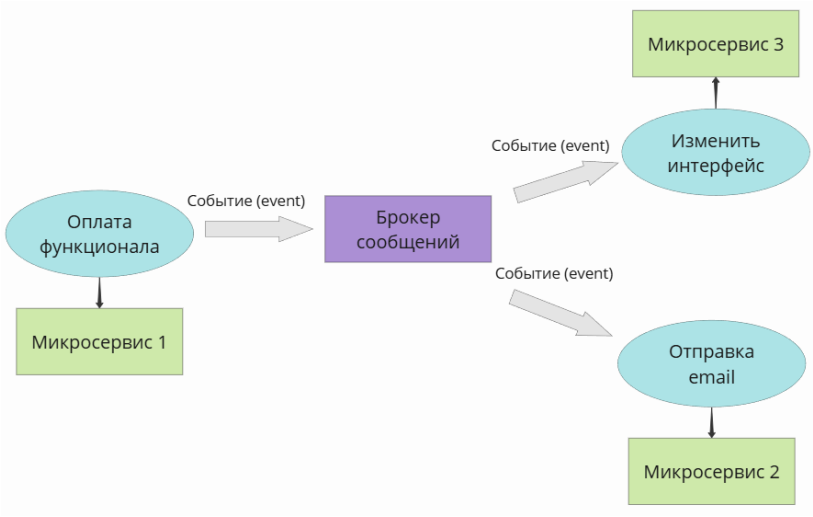


Рисунок 3 – Схема работы паттерна проектирования EDA.

2.1.3 Архитектура клиентской части приложения

Существуют различные подходы к проектированию клиентской части приложения, рассмотрим некоторые из них:

– Server Side Rendering (SSR) – это технология, с помощью которой генерация веб-страницы осуществляется на стороне сервера и отправляется на клиент в виде HTML-код [31]. Клиенту достаточно просто отобразить полученный код.

– Client-Side Rendering (CSR) – страница генерируется на стороне клиента. Frontend, используя JavaScript или другой подобный язык, обращается на сервер посредством АПИ эндпоинтов, с которых получает данные (чаще всего используется json), далее рендерит страницу [32].

– Single-Page Application (SPA) [33] – полная генерация страницы осуществляется один раз, дальнейшая навигация происходит без перезагрузки страницы, а информация изменяется посредством обновления HTML разметки с помощью JavaScript.

– Multi-Page Application (MPA) – противопоставляется SPA: каждая новая ссылка ведет к перезагрузке страницы и рендеринга HTML разметки целиком.

Наглядно ознакомиться с логикой работы SPA и MPA можно ознакомиться на рисунке 4.

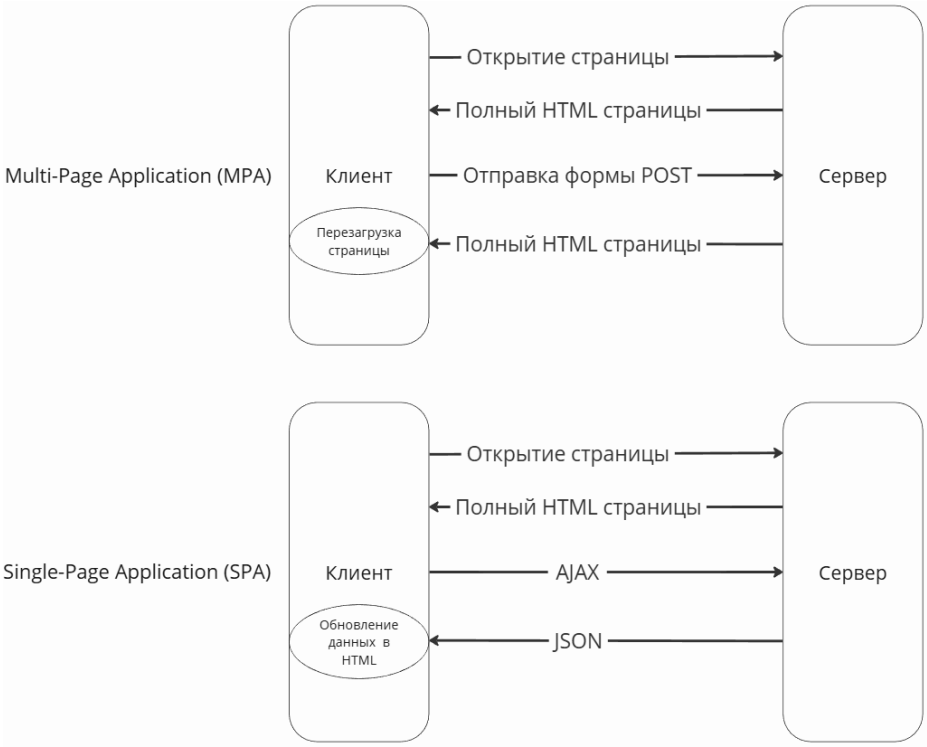


Рисунок 4 – Сравнение логики работы MPA и SPA

Современные frontend фреймворки, такие как: Vue.js, React, Angular – позволяют использовать лучшие практики из каждого подхода. Так, страница может быть разбита на различные части: переиспользуемые и те, которые необходимо обновлять. Наиболее частое применение – выделение верхней и нижней частей сайта, которые остаются неизменными на всех страницах, в отдельные компоненты, таким образом, что frontend не будет их генерировать при переходе на соседнюю страницу сайта. Для наглядности представлен рисунок 5, на котором представлены шапка и нижняя часть сайта, как постоянные элементы сайта, которые не изменяются при переходе на другую страницу. А тело страницы будет меняться при переходе по вкладкам.

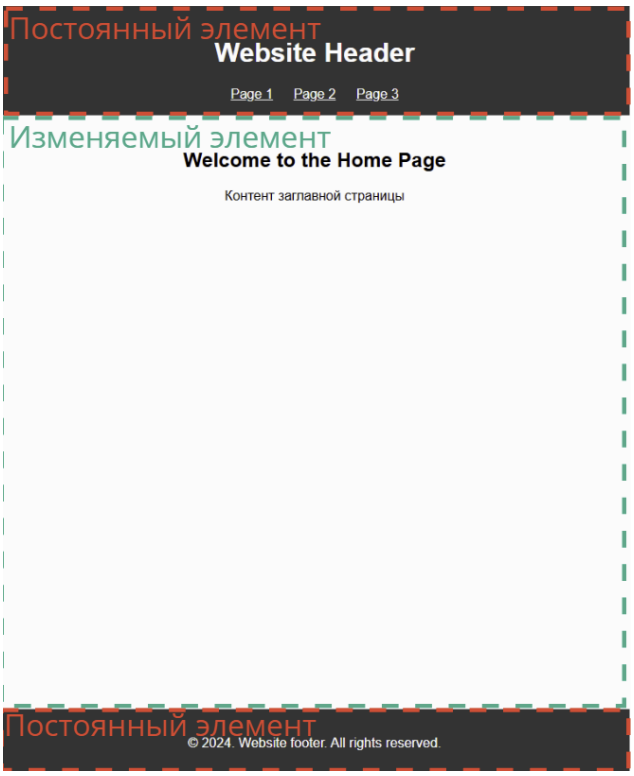


Рисунок 5 – Пример постоянных и изменяемых элементов страницы.

Именно такой подход – наиболее популярен в современной разработке. Также следует упомянуть, что frontend, в рамках работы, будет получать необработанные данные в формате json с сервера, а уже потом рендерить страницу для пользователя. Таким образом разработчики снижают размер передаваемой информации с сервера клиенту, ускоряя работу и снижая нагрузку на backend.

2.2 Технологический стек приложения

2.2.1 Технологический стек серверной части приложения

Для разработки быстрой, отказоустойчивой и безопасной серверной части приложения необходимо подобрать подходящие инструменты и технологии.

Первым этапом, из аналогов был выбран веб-сервер, который будет обрабатывать запросы пользователя. Наиболее популярными являются

NGINX [35] и Apache [36-37]. Ниже представлена таблица 1 сравнения этих двух веб-серверов.

Таблица 1 – Сравнение веб-серверов NGINX и Apache.

Характеристика	**NGINX**	**Apache**
Тип	Веб-сервер и прокси-сервер	Веб-сервер
Архитектура	Асинхронная	Процесс-ориентированная
Производительность	Высокая	Высокая, менее эффективна при множественных запросах
Ресурсы	За счет внутренних алгоритмов обеспечивает более эффективное использование CPU	Менее эффективное использование CPU
Конфигурация	Гибкая, простая и понятная	Гибкая, но довольно сложна для восприятия
Модульность/расширения	Присутствуют	Присутствуют
Поддержка	Огромное сообщество, стабильно обновляется	Огромное сообщество, обновляется реже

Взвесив все плюсы и минусы этих двух самых популярных веб-серверов, было принято решение остановиться на NGINX, так как он более современный, прост в установке и настройке, более эффективно справляется с нагрузками и имеет «из коробки» прокси-сервер, который выступает посредником между клиентом и серверами. Фактически, как веб-сервер, он

обслуживает запросы пользователя, возвращая статические файлы, выполняет запросы к серверным приложениям. Как прокси-сервер, он перенаправляет запросы на другие сервера, в зависимости от реализованной логики, что позволяет балансировать нагрузку и выполнять роль обратного прокси для защиты.

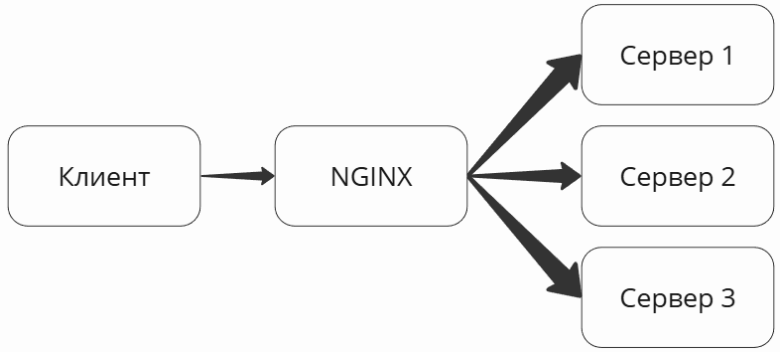


Рисунок 6 – Пример работы балансировщика нагрузки

В качество основной части приложения использовался язык программирования Python [38]. Для реализации серверной веб-части использовали фреймворк Django [39]. Он имеет множество преимуществ перед такими фреймфорками как Flask [40] или AioHTTP [41]. Более подробно с преимуществами и недостатками можно ознакомиться в приложении Г.

Django содержит и свои преимущества, например, встроенный ORM и административный интерфейс, и недостатки: например, плохая поддержка асинхронности. Тем более, у команды разработки был довольно большой опыт работы с этим фрейморком, так что было принято решение остановиться на нем. Кроме Django в проекте применялся Django Rest Framework [42] – фреймворк для создания API на основе Django.

Для реализации системы очередей была выбрана Celery [43], которая представляет собой распределенную асинхронную очередь задач, которая работает в связке с одним из нескольких на выбор брокеров сообщений. Это очередь задач, ориентированная на обработку в реальном времени, а также поддерживающая планирование задач [44]. На сегодняшний день большая

часть проектов используют Redis [45], либо RabbitMQ [46] в качестве брокеров сообщений. Вот несколько ключевых отличий двух инструментов:

Тип данных и модель хранения:

– Redis: представляет собой хранилище данных типа ключ-значение. Он широко используется для кэширования данных, хранения сессий, и обработки структурированных данных. Redis поддерживает различные типы данных, такие как строки, списки, множества и хэши.

– RabbitMQ: является брокером сообщений, который обеспечивает механизм для отправки, получения и обработки сообщений. Он реализует протокол AMQP (Advanced Message Queuing Protocol) и предназначен для построения распределенных систем с использованием очередей сообщений.

Модель коммуникации:

– Redis: предоставляет средства для взаимодействия между приложениями через сетевой интерфейс. Он может использоваться для обмена данными между компонентами системы, но не предоставляет встроенных механизмов для организации сложных сообщений и событий.

– RabbitMQ: предоставляет мощный механизм обмена сообщениями. Он поддерживает различные типы обмена сообщениями, такие как прямой, тематический, веерный и др.

Подход к распределению данных:

– Redis: обычно используется как централизованное хранилище данных, к которому имеют доступ различные компоненты системы. Он может быть использован в режиме мастер-слейв для обеспечения отказоустойчивости.

– RabbitMQ: является распределенным брокером сообщений, который может быть развернут в кластере для обеспечения отказоустойчивости и масштабируемости.

Гарантии доставки сообщений:

– Redis: предоставляет ограниченные гарантии доставки сообщений. Он обеспечивает сохранение данных в памяти, но при перезапуске данные могут быть потеряны.

– RabbitMQ: обеспечивает надежную доставку сообщений, поддерживая различные уровни гарантий, такие как «at most once» (не более одного раза), «at least once» (не менее одного раза) и «exactly once» (точно один раз).

Для проекта выбран Redis. Redis это хранилище данных типа «ключ – значение», поддерживающая разные типы и структуры данных, например: списки, наборы, хэши, массивы, состоящие из битов и прочие. К моменту интеграции связки Celery и Redis у команды уже был достаточно большой опыт работы именно с этой связкой, что гарантировало нам простоту, скорость и надежность интеграции.

Для Celery разработано расширение Celery Beat, представляющий собой планировщик задач (далее – таск), который может запускать различные таски в автоматическом режиме и через определенное время, либо в конкретное время [47].

В качестве системы управления базами данных (СУБД) использовался PostgreSQL [48]. Также рассматривали аналоги, например, классический MySQL [49], с результатами сравнения можете ознакомиться в приложении Д.

Хотя MySQL все также популярна в сообществе разработчиков, на рисунке 7 наглядна видна тенденция к стагнации, которая вызвана очевидным и не очень недостаткам [50].

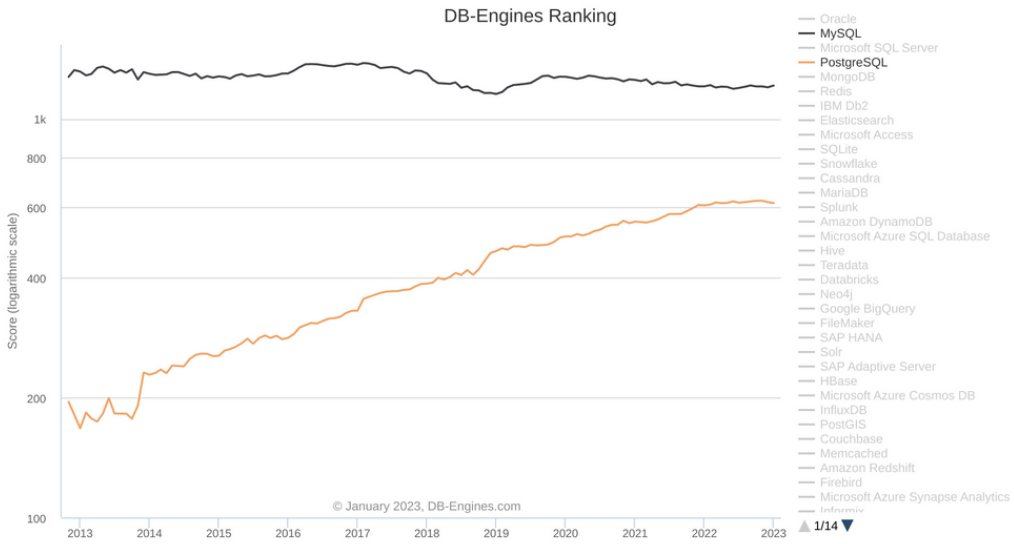


Рисунок 7 – Демонстрация востребованности баз данных

Подводя итог, технологический стек северной части приложения выглядит следующим образом: Docker, Nginx, Python (Django, Django Rest Framework, Celery-beat, websockets), Celery, Redis, PostgreSQL.

2.2.2 Технический стек клиентской части приложения. Схема взаимодействия

На сегодняшний день существует огромное многообразие различных технологий, позволяющих реализовывать клиентскую часть приложений [51]. В подавляющем большинстве случаев используется JavaScript, HTML и CSS. На рынке присутствует довольно много фреймворков-надстроек над JavaScript. Для ведения проекта необходимо было выделить ключевые особенности каждого из них, чтобы понять, какой мы будем использовать. С результатами сравнений можно ознакомиться в таблице 3.

Таблица 3 – Сравнение популярных Frontend фреймворков

Характеристика	**Vue.js**	**React**	**Angular**
Язык программирования	JavaScript	JavaScript	JavaScript (TypeScript)
Рендеринг	Виртуальный DOM	Виртуальный DOM	Виртуальный DOM
Компонентный подход	Да	Да	Да
Реактивность	Да	Да	Да
Обучение и документация	Отличная	Отличная	Хорошая
Инструменты разработки	Vue CLI, Vue Devtools	Create React App, React Devtools	Angular CLI, Angular Devtools
Производительность	Высокая	Высокая	Высокая

Для проекта использовался Vue.JS [52] по нескольким причинам: его гибкость и масштабируемость, высокая производительность, активное сообщество, простота и интуитивность. Кроме того, Vue.JS позволяет в очень понятной форме делить составляющие проекта, позволяя переиспользовать кодовую базу.

В качестве удобного инструмента для деплоя приложения на сервер было принято решение использовать Docker – это платформа для разработки и доставки и запуска приложений с использованием контейнеризации. Данная технология позволяет легко создавать унифицированную среду для разработки, тестирования и продакшена, что минимизирует проблемы совместимости и обеспечивает стабильность приложения на всех этапах жизненного цикла. Благодаря контейнеризации, развертывание и

масштабирование приложения становятся простыми и быстрыми, а управление зависимостями и обновлениями упрощается.

Для реализации клиентской части приложения используется Vue.JS, обеспечивающий высокую производительность и удобство разработки. Серверная часть построена на базе Django с использованием Django Rest Framework (DRF) для создания REST API. Взаимодействие между компонентами осуществляется через Nginx, выполняющий функции обратного прокси-сервера, и Gunicorn, выступающий в качестве WSGI-сервера. Важными элементами архитектуры являются также системы кэширования и хранения данных, представленные Redis и PostgreSQL соответственно. Для асинхронной обработки задач используются Celery и Celery Beat, а для мониторинга состояния системы – Observer. Внешние данные получаются через API Binance, и вся архитектура управляется администратором через Telegram.

Выбранный стек полностью бесплатен, абсолютно каждый инструмент имеет подробную документацию и активное сообществе. С общей схемой работы приложения можно ознакомиться в приложении В.

Представленные технологии в сочетании с выбранными технологиями обеспечивают стабильность и масштабируемость web-приложения. Контейнеризация позволяет создать уникальную среду разработки, тестирования и деплоя, что значительно снижает количество ошибок и упрощает управление зависимостями.

3 РАЗРАБОТКА WEB-СЕРВИСА ДЛЯ ВНУТРЕДНЕВНЫХ ТРЕЙДЕРОВ

3.1 Технический стек клиентской части приложения. Схема взаимодействия

После изучения документации к API Binance [52], просмотра примеров запросов и ответов, было принято решение начинать проводить пробную интеграцию и начинать непосредственно разрабатывать web-приложение для торговли на бирже Binance.

Перед началом работы необходимо понять, чем отличается HTTP запросы от Websockets (Binance поддерживает оба варианта обмена данными).

HTTP и Websockets являются протоколами передачи данных для взаимодействия между клиентом и сервером. HTTP является протоколом запрос-ответ, который используется для передачи статических данных между клиентом и сервером. Websockets же является протоколом двустороннего взаимодействия, который позволяет клиенту и серверу обмениваться динамическими данными в реальном времени [53-54]. В отличие от HTTP, Websockets поддерживает постоянное соединение между клиентом и сервером, что позволяет им обмениваться данными в реальном времени.

HTTP использует протокол TCP для передачи данных, а Websockets использует протокол TCP или UDP. HTTP использует простой текстовый формат для передачи данных, а Websockets использует бинарный формат для передачи данных [55-56]. Более подробное сравнение далее по тексту.

Типы:

- Ключевое отличие HTTP от Websockets в том, что все запросы HTTP однонаправленные, то есть, соединение закрывается в тот же момент, когда на первичный запрос получен ответ от сервера.
- Websockets, в свою очередь, двунаправленный, это значит, что он может одновременно и получать данные, и возвращать.

Скорость работы:

– HTTP – использует протокол запрос-ответ, поэтому для передачи данных между клиентом и сервером необходимо отправлять запросы и получать ответы. Таким образом, существенно возрастает нагрузка на сеть и железо.

– Websockets использует протокол двустороннего взаимодействия, поэтому данные могут быть переданы между клиентом и сервером без необходимости отправлять запросы и получать ответы. Это позволяет передавать данные быстрее чем в HTTP. Кроме того, нет необходимости каждый раз отправлять XMLHttpRequest, что снижает объем отправляемых данных.

Безопасность:

– HTTP имеет более защищенную и рекомендованную версию https. Это безопасный протокол для передачи данных. Используется криптографическое SSL и TSL шифрование

– Websocket’ы имеют аналогичную систему защиты, доступную по запросу на wss.

Наглядно принцип работы продемонстрирован на рисунках 8 и 9

Рисунок 8 – Демонстрация работы HTTP/HTTPS

Рисунок 9 – Демонстрация работы WS/WSS

Для проекта использовался Websocket, как наиболее предпочтительный протокол общения с серверной частью. Принцип работы протокола позволит сократить время общения «клиент-сервер-клиент» и даст возможность уменьшить нагрузку на аппаратную часть.

Кроме выбора между типом подключения, необходимо было определиться с подходами к написанию кода. Асинхронный подход к программированию представляет собой парадигму, в которой процессы и задачи выполняются независимо друг от друга. Это позволяет программистам писать более эффективный и гибкий код, который может быть использован для решения различных задач. Основные преимущества асинхронного подхода к программированию включают в себя улучшение производительности, уменьшение задержек и простоту использования. асинхронный подход позволяет программистам писать более эффективный и гибкий код, который может быть использован для решения различных задач. Кроме того, асинхронный подход позволяет программистам избегать проблем, связанных с блокировкой потоков и позволяет им более эффективно использовать ресурсы процессора. Асинхронный подход к программированию предоставляет множество преимуществ для программистов, включая

улучшение производительности, уменьшение задержек и простоту использования [57].

При синхронном исполнении кода все задачи попадают в список задач для воспроизведения и начнут выполняться только когда подойдет их очередь. Асинхронный подход к написанию кода позволяет запускать участки кода независимо друг от друга, что положительно сказывается на времени полного выполнения кода. На рисунках 10 и 11 схематично представлена работа синхронного и асинхронного кода.

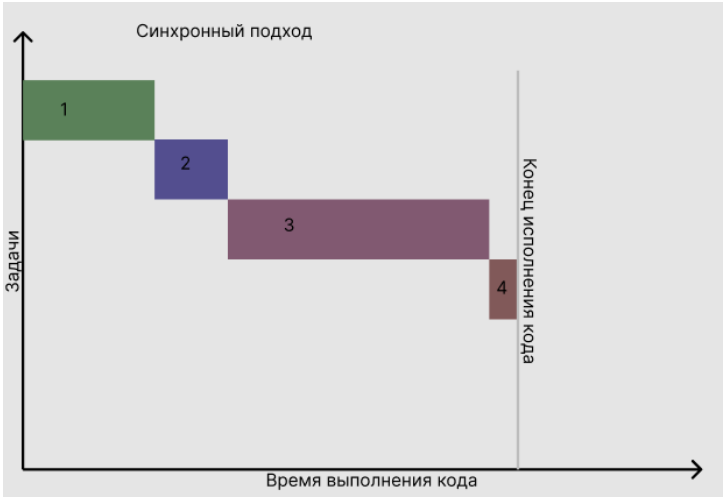


Рисунок 10 – Схема времени выполнения синхронного кода

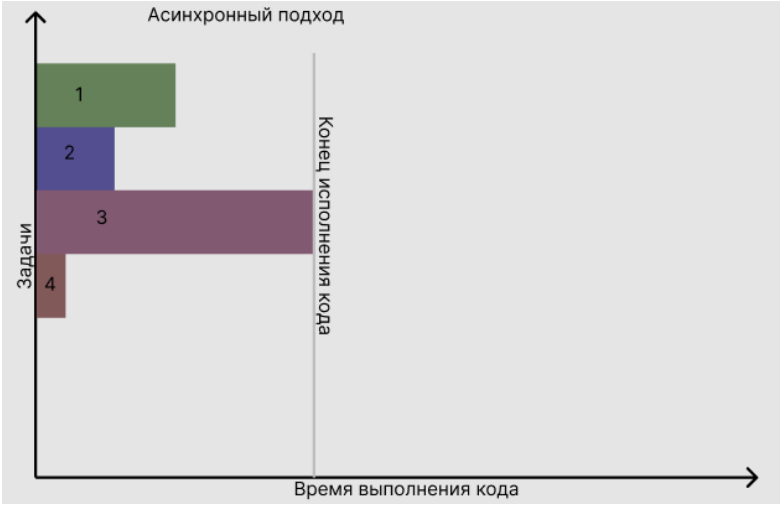


Рисунок 11 – Схема выполнения асинхронного кода

3.2 Реализация MVP

3.2.1 Обновление графиков

Реализацию функционала было решено начать получением первичных, необработанных данных с Binance, используя websockets и API, предоставляемым Binance. Изначально была реализован алгоритм, устанавливающий соединение с Binance и получающий весь доступный orderbook, вычленяющий нужные данные. Далее данные визуализировались с помощью библиотеки Mathplotlib [58]. В результате удалось получить, обработать и визуализировать данные, оставалась проблема, что данные статичны (график не обновляется при получении новых значений). На рисунке 12 представлена часть кода, отвечающая за построение графиков на основе данных, полученных с API Binance.

В качестве задачи и итогового продукта второй итерации было решено дорабатывать графическое отображение графика. Как итог: график получил возможность ежесекундно обновляться. Средняя скорость обновления графика: 0.5 секунды. На рисунке 13 представлен продукт второй итерации: автоматически обновляющийся график на фоне текущего курса на бирже Binance.

```python
async def formatter(order_book: dict) -> dict:
    """
    Get raw data and format it to 2 lists 'bids, and 'asks'
    :param order_book: must be dict
    :return: list(bids), list(asks)
    """
    formatted_bids = [[float(price), float(quantity)] for price, quantity in order_book['bids']]
    formatted_asks = [[float(price), float(quantity)] for price, quantity in order_book['asks']]
    return {'asks': formatted_asks,
            'bids': formatted_bids}


async def accumulate_data(formatted_bids: list, formatted_asks: list) -> dict:
    """
    Accumulating data from list to build right graph
    :param formatted_bids:
    :param formatted_asks:
    :return:
    """
    accumulate_quantity_bids = accumulate(
        [quantity for price, quantity in formatted_bids],
        operator.add
    )
    accumulate_quantity_asks = accumulate(
        [quantity for price, quantity in formatted_asks],
        operator.add
    )
    return {'accumulate_quantity_bids': accumulate_quantity_bids,
            'accumulate_quantity_asks': accumulate_quantity_asks}


async def get_order_book(symbol: str) -> dict:
    """
    using binance api get order book
    :param symbol: str like 'BTCUSDT'
    :return: dict of values from selected pairs of values
    """
    return await client.get_order_book(symbol=symbol, limit=5000)
```

Рисунок 12 – Участок кода, отвечающий за построение графиков по данным с API Binance.

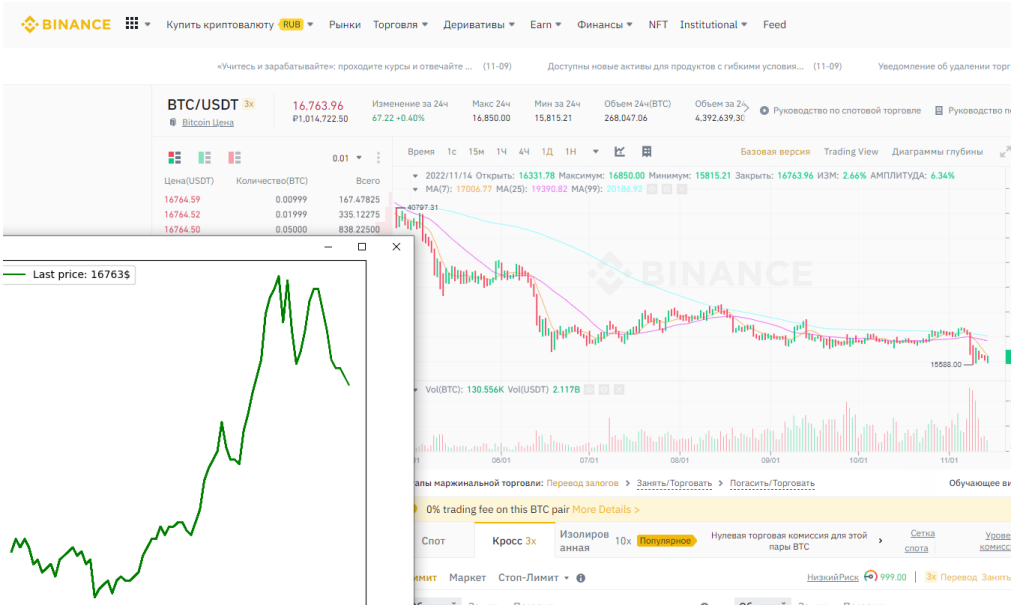


Рисунок 13 – Динамический график стоимости BTC/USDT.

3.2.2 Пользовательский API

В рамках третьей итерации была поставлена цель связать пользователей в личном кабинете нашего сервиса непосредственно с API интерфейсом биржи Binance. Как итог итерации: дали пользователю возможность создавать и настраивать API ключи, создавать разные виды кошельков, не заходя на сам сайт Binance, проводить транзакции между счетами и закрывать заявки. Чуть более подробно по каждому пункту ниже.

– Создавать и настраивать API-ключи:

прежде всего, следует отметить, что, как говорили ранее, используется Websocket соединение. Соединение устанавливается один раз, далее через него есть возможность отправлять запросы на сервер и получать ответы. Для всех запросов корневая структура json’а в теле сообщения одинаковая:

```
{
  "action": "Имя запрашиваемого действия",
  "payload": {
    // Обязательные поля для данного действия
  }
```

Структура ответов тоже одинаковая:

```
{
  "action": "Имя действия, на запрос по которому отправлен ответ",
  "payload": {
    // Данные ответа
  },
  "status": 100
  // Число (int) Могут быть статусы 100, 200, 201, 204}
}
```

Для всех типов ошибок структура одинаковая, но отличается от обычных («успешных») ответов:

```
{
    "action": "Имя действия, на запрос по которому отправлен ответ",
    "payload": ["Текст ошибки 1", "Текст ошибки 2", ..., "Текст ошибки n"]
    "status": 500 // Число (int) Могут быть статусы 400, 401, 404, 500
}
```

– Для получения API ключа необходимо направить:

```
 {
  "Action" : "create_API", "payload": values
}
```

По ключу payload передаются: id пользователя; название ключа, которое будет отображаться в личном кабинете; доступ, в котором можно указать спотовый это или фьючерсный кошелек; а также IP адрес пользователя.

В случае успешного выполнения запроса сервер возвращает подтверждение, в котором указывает название операции (в данном случае, create_API), название ключа и статус со значением 100. В случае неудачи сервер явно обозначит это, поставив значение статуса равным одному из значений, перечисленных выше, а в данных («payload») укажет причину неудачи (например, «неправильный код Google Authenticator» или «This field is required» с указанием незаполненного поля).

После имплементации указанного функционала, web-приложение стало способным:

1) Создавать кошельки для фьючерсной торговли; создавать кошельки для спотовой торговли. Создание API для фьючерсного или спотового кошелька происходит по стандартному запросу

‘Action’: ‘create_API’, и в теле запроса явно указывается тип кошелька путем подстановки булиевого значения. Пример такого запроса:

```
{
  ‘permissions’: {‘spot’: True/False, ‘futers’: True/False}
};
```

Готовый интерфейс получившегося инструмента на рисунке 14.

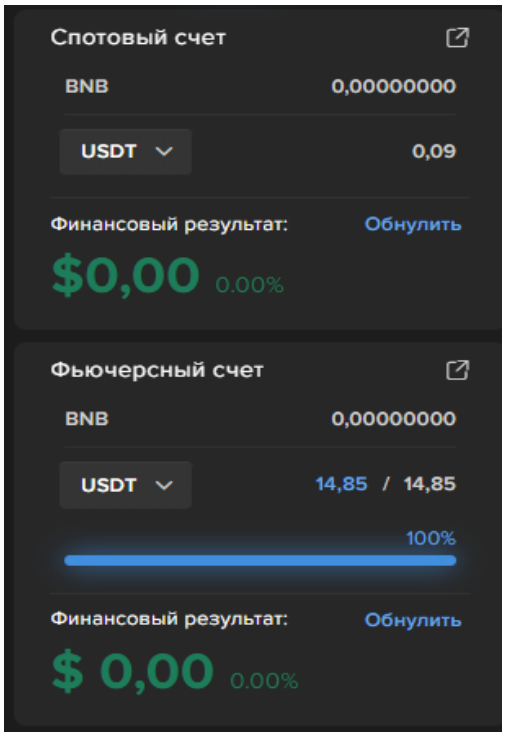


Рисунок 14 – Интерфейс кошельков для спотовой и фьючерсной торговли

2) Изменять ключ (action: ‘change_API_name’);

3) Изменять настройки API ключа (action: ‘manage_API’);

4) Удалять API ключ (action: ‘delete_API’);

5) Получать список привязанных API ключей (action: ‘API_keys’).

Готовый интерфейс инструмента для создания API на рисунке 15.

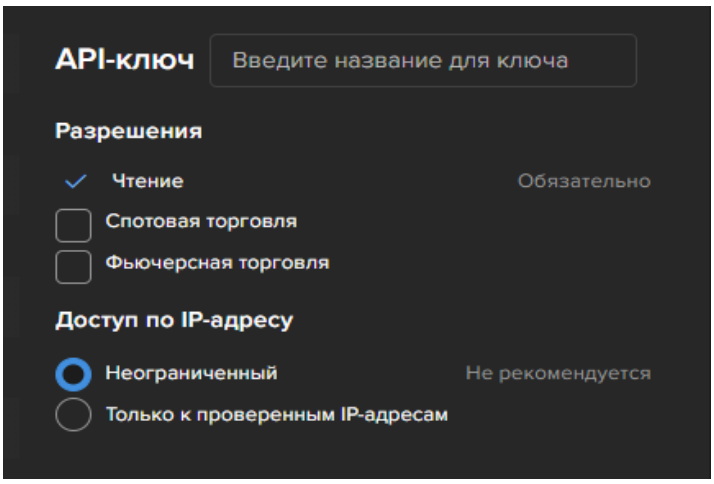


Рисунок 15 – Интерфейс созданияAPI-ключа

– Переводить средства между созданными кошельками;

Сразу же после открытии страницы, на которой осуществляется управление переводами между кошельками, происходит запрос на сервер. В ответе указана доступный баланс, используемые валюты, адрес кошелька, минимальная сумма вывода и комиссия за операцию. Сделано это из соображений упрощения изменения настроек в последующем (например, появится необходимость установить другие значения комиссионного сбора или минимальной суммы для перевода).

После нажатия кнопки перевод по факту не осуществляется, а создается на него заявка в базе. Заявку необходимо подтвердить путем отправки СМС на номер телефона пользователя. Сервер проверяет введенный пользователям код с эталоном, возвращает либо ошибку, либо подтверждение операции. Кроме СМС подтверждение требуется проверка, что сумма перевода больше или равна минимальной, а также, что сумма перевода есть на счету трейдера. По факту успешного подтверждения СМС кода и выполнения условий происходит перевод средств, а сервер возвращает status: 200, что означает успешное выполнение операции.

С готовым интерфейсом разработанного виджета можно ознакомиться на рисунке 16.

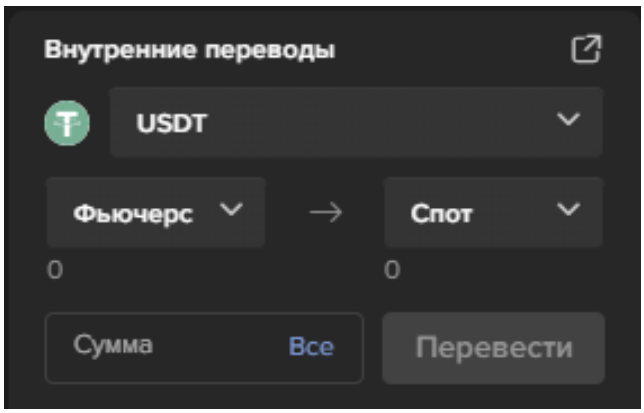


Рисунок 16 – Перевод средств между созданными кошельками

– Закрывать позиций и лимитные заявки, открытые трейдерами.

Web-приложение предполагает, что конечный пользователь открывать позиции и создавать лимитные заявки будет через торговый терминал, так как именно торговый терминал является основным рабочим инструментом трейдера. Открытие заявок происходит путем отправки POST запроса, в теле которого указываются обязательные параметры. Дополнительные параметры необходимо указывать в зависимости от условий запроса, например, для стоп-лосс или тейк-профит заявки необходимо дополнительно указать *stopPrice* и *trailingDelta*.

Разработанное web-приложение дает возможность трейдерам закрывать лимитные заявки до срока их исполнения (при открытии заявки в нем можно указать на то, что эта заявка является лимитной) через личный кабинет трейдера.

В документации API Binance указана возможность закрытия любой заявки (не важно, лимитная она или нет) путем отправки DELETE запроса с указанием реквизитов заявки (*orderId* или *origClientOrderId*). Можно закрыть все лимитные заявки, путем направления запроса, где в параметре *«limit_orders»* перечисляются все лимитные заявки, а в качестве ответа сервер возвращает json, где в параметре *«closed_limit_orders»* перечисляются все лимитные заявки.

Готовый интерфейс виджета представлен на рисунке 17.

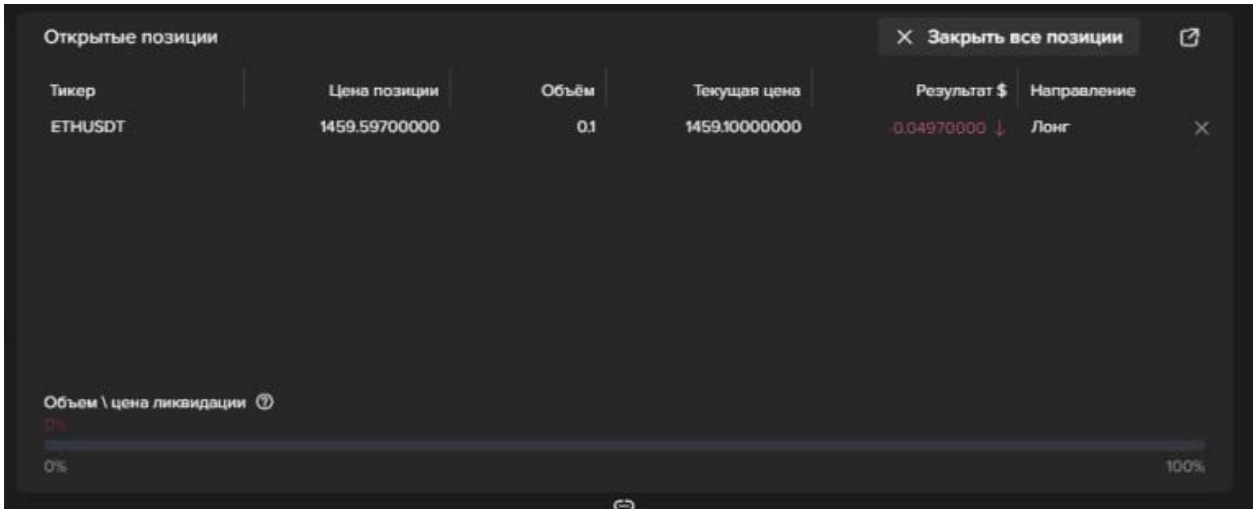


Рисунок 17 – Закрытие позиций и лимитные заявки, открытые трейдерами

3.3 Профиль пользователя, система бонусов

3.3.1 Регистрация, авторизация и аутентификация, шифрование

После внутренней апробации реализованного функционала, было принято решение добавлять полноценные сервисы для взаимодействия непосредственно с пользователем и личным кабинетом. Регистрация реализована путем создания дополнительного приложения в среде Django, которое сделано только для того, чтобы отдельно вынести модели (структуру базы данных). Django предлагает свои модели пользователей «из коробки», но они не совсем подходят нам по содержанию. Например, нам необходимо завести дополнительные поля для номера телефона, для имени в приложении Discord для связи и т. д. Существует несколько способов изменить содержание стандартной модели:

– простое расширение модели (proxy): новые таблицы в базе данных не создаются. Чаще всего используют, чтобы изменить поведение внутри существующей модели, без затрагивание существующей схемы базы данных;

– использование связи OneToOneField: это стратегия с использованием дополнительной обычный модели Django со своей таблицей в базе данных, которая связана пользователем стандартной модели через связь OneToOneField. Эту стратегию, традиционно используют, чтобы хранить дополнительную информацию, которая не связана с процессом аутентификации (например, телефонный номер или другие данные).

– расширение AbstractBaseUser: при использовании данного метода создается новая модель пользователя, которая наследуется от AbstractBaseUser. Чаще всего этот метод используют, когда есть специфические требования в отношении процесса аутентификации. Способ сложный в работе и часто дает сбои;

– расширение AbstractUser эту стратегию используют, когда сам процесс аутентификации Django полностью удовлетворяет и нет необходимости менять его. Но, вместе с тем, есть потребность добавить некоторую дополнительную информацию непосредственно в модели пользователя, без необходимости создавать дополнительный класс.

Исходя из задач проекта, был реализован метод OneToOneField, была создана дополнительная модель, где заложены все необходимые для работы поля и связали методом OneToOneField со стандартной моделью User. Таким образом, удалось расширить поля у пользователей, не изменяя логику работы всех остальных компонентов.

В основе нашего проекта лежит фреймворк Django. Он позволяет использовать несколько видов авторизации и передачи данных о пользователе между страницами:

– авторизация на основе сессий (Session-based authentication): Django использует механизм сессий для управления аутентификацией пользователей. После успешной аутентификации Django создает уникальный идентификатор сессии для пользователя, который сохраняется на стороне сервера или может быть сохранен в куки на стороне клиента. При каждом запросе сервер

проверяет, действительна ли сессия и разрешает или запрещает доступ к ресурсам в зависимости от этого;

– авторизация на основе токенов (Token-based authentication): Django также поддерживает авторизацию на основе токенов. В этом случае, после успешной аутентификации, сервер выдает клиенту уникальный токен, который клиент должен предоставлять с каждым запросом для подтверждения своей авторизации. Токены могут быть переданы через HTTP-заголовок или параметры URL. Django предоставляет готовые средства для генерации и проверки токенов;

– авторизация на основе JSON Web Tokens (JWT): JWT – это открытый стандарт (RFC 7519) для безопасной передачи информации между двумя сторонами в формате JSON. Django имеет множество сторонних библиотек для работы с JWT, которые обеспечивают удобные методы генерации и проверки токенов. Авторизация с использованием JWT позволяет создавать токены, содержащие дополнительную информацию о пользователе или о других связанных с ним данных;

– авторизация с помощью сторонних сервисов: Django также позволяет использовать сторонние сервисы авторизации, такие как OAuth или OpenID, для аутентификации пользователей. С помощью этих сервисов пользователи могут войти в ваше приложение, используя свои учетные данные от сервисов, таких как Google, Facebook или Twitter. Django предоставляет удобные библиотеки и инструменты для интеграции с различными сторонними сервисами.

Для нашего проекта мы выбрали JWT, так как он имеет несколько преимуществ [59]:

– без состояния (Stateless): JWT является механизмом «без состояния», что означает, что сервер не хранит информацию об аутентификации пользователя. Вся необходимая информация, такая как данные пользователя и разрешения, зашифрована в тексте токена. Самый

очевидный плюс данного пункта: возможность масштабирования, так как для верификации токена можно проверить без обращения к централизированному хранилищу данных и состояний;

– безопасность: JWT подписывается сервером с использованием секретного ключа или приватного ключа RSA. Когда клиент отправляет токен на сервер, сервер может проверить его подлинность, расшифровать информацию о пользователе и принять решение о предоставлении доступа. Поскольку токены подписываются, они не могут быть подделаны или изменены злоумышленником без знания секретного ключа;

– переносимость и расширяемость: JWT является форматом данных в формате JSON, что делает его удобным для использования в различных средах и платформах. Токены можно передавать через HTTP-заголовки, параметры URL или в теле запроса. Таким образом, мы сделаем некоторый задел и оставим возможность интегрировать и использовать эту систему не только с ПК, но и с мобильных устройств (возможно, когда-то мы задумаемся о разработке собственного приложения под iOS или Android). Также именно JWT традиционно используют для разработки SPA (Single Page Application «одностраничных» сайтов);

– персонализация и хранение информации: JWT может содержать дополнительную информацию о пользователе, которая быть передана и использована в каких-либо целях. В токене можно включить ID пользователя, разрешения или другие пользовательские данные, которые можно использовать на стороне сервера без необходимости дополнительных запросов к базе данных. Данное преимущество JWT потенциально поможет нам не только снизить нагрузку на сервера и базу данных, но и оптимизирует некоторые запросы;

– легкая интеграция со сторонними сервисами: JWT широко используется в сторонних сервисах аутентификации, таких как Google, Facebook и другие. Это означает, что вы можете использовать JWT для

интеграции с этими сервисами и позволить пользователям аутентифицироваться в вашем приложении с использованием своих учетных данных от других платформ;

– обширное сообщество: JWT – это один из стандартных видов авторизации, которые используют многие разработчики не первый год по всему миру. Таким образом, база знаний, находящаяся в публичном доступе, исчерпывающая и позволяет найти ответы на многие вопросы.

Касательно логики работы: JWT используется для авторизации, в самом токене мы шифруем уникальный ID пользователя, далее все сервисы, которым передается этот токен, сверяют ID и необходимые им данные в записи пользователя в базе данных (права, роль и т.д.) и в зависимости от этого обрабатывают запрос. Схема представлена на рисунке 18.

Рисунок 18 – Схема взаимодействия с JWT токеном

JSON, который мы отправляем на сервис авторизации имеет стандартный для JWT авторизации формат:

```
{
"email": "admin@example.com", "password": "admin"
}
```

В ответ сервер возвращает нам непосредственно JSON Web Token формата:

```
{
  "access_token": "TOKEN", "refresh_token": "RTOKEN", "user: {"uid": SHA256(id)
}
```

На данном этапе нам нужна именно такая структура JWT: мы получает два уникальных токена и зашифрованный уникальный идентификатор пользоваля. Все данные о пользователе хранятся в таблице ClientAccounts, которая содержит связь API в таблице с ключами пользователя, secret, login, password, phone – все поля зашифрованы переписанным и улучшенным методом шифрования, который предлагает Django (*Django.contrib.auth.hashers.make_password/check_password* – расположение стандартных функция создания и проверки пароля).

По умолчанию, Django шифрует все пароли по алгоритму PBKDF2 (Password-Based Key Derivation Function 2). Он представляет собой криптографический стойкую функцию, в основе которой лежит множество итераций хэширования [60]. Алгоритм требует нескольких составляющих:

- соль (salt) – это некий псевдослучайный набор символов, который конкатенируется с паролем перед хэшированием, что гарантирует уникальность кэша даже в случае, если два пользователя установят одинаковые пароли;
- итерации – алгоритм по умолчанию применяется несколько раз, что существенно увеличивает устойчивость брутфорсу;
- секретный ключ – заданный набор символов, который также добавляется к паролю перед хэшированием.

Однако, разрабатываемое web-приложение по умолчанию предполагает крайне высокие требования к безопасности из-за взаимодействия с денежными средствами, так что было принято решение использовать алгоритм Argon2, разработанный Бирюковым, Дину и Ховратовичем из университета Люксембурга в 2015 году [61]. Основной упор этого метода шифрования – защита от атак методом перебора. Он обладает очень гибкими конфигурациями, что позволяет подобрать оптимальный баланс между скоростью вычислений и их устойчивости.

Реализовав функционал смены пароля, номера телефона и почты, пользователю предоставлена возможность самостоятельно решать проблемы

с утратой почты или телефона, что помогает в дальнейшем снизить нагрузку на команду проекта.

3.3.2 Сервис сбора статистических данных пользователя

Следующим этапом развития продукта стала реализация системы cashback’а, аналогично системам, которые предлагают многие банки.

Для расчета количества начисленных бонусов и дальнейшей выплаты и распределения их необходимо реализовать систему сбора статистических данных по всем сделкам пользователей. Поскольку cashback рассчитывается в процентном соотношении от комиссии, уплаченной пользователем бирже за каждую сделку, необходимо получать с биржи информацию по каждой сделке и размеру комиссии за нее.

API Binance предоставляет очень много различных инструментов, которые, кроме того, имеют обширную документацию, что позволяет без каких-либо проблем интегрировать нужный инструмент и начать его использования без необходимости доделывать функции. Не мало важно отметить, что Binance время от времени обновляет функционал своего API, расширяя его возможности и оптимизируя существующие алгоритмы.

Используя секцию «Payload», доступную по URL: *«/api/v3/userDataStream»*, который, в свою очередь, требует установления соединения по web-socket мы можешь направлять различные запросы на API, и получать информацию о торговле пользователя через наш сервис (разумеется, для получения той или иной информации необходимо перед установкой соединения передать зашифрованный ключ, чтобы подтвердить права на доступ к функциям этого endpoint).

Можно получить достаточно много разных данных по каждой сделке пользователя, например: дату, тикер, объем в лотах, PNL в USDT, PNL без комиссии в USDT, саму комиссию, валюту комиссии, время входа и выхода и так далее. Для наших же целей достаточно получать время входа и выхода из

сделки, торговую пару, объем, площадку и цену валют на момент входа и выхода из сделки. Используя эти данные, можно рассчитать количество монет для начисления, используя формулу, представленную на рисунке 19.

Рисунок 19 – Правила начисления бонусов.

Из-за того, что ежедневно пользователи могут совершать очень большое количество сделок (напомним, что большое количество сделок – одна из особенностей внутридневного трейдинга) мы приняли решение собирать их постепенно, на протяжении всего дня, разделив его на отрезки с промежутками в 4 часа между ними.

Используя эти инструменты, мы подготовили несколько таск, которые опрашивают биржу каждый 4 часа и собирают необходимые данные, схема работы представлена на рисунке 20.

Рисунок 20 – Схема времени запуска ежедневных задач.

Так появилась возможность ежедневно обрабатывать очень большое количество данных, не загружая сервера. Кроме того, повысили отказоустойчивость, так как, если что-то пойдет не так в ходе выгрузки или

обработки данных, будет известно в каком конкретно промежутке произошел сбой и будет возможность проанализировать, что пошло не по сценарию. После получения этих данных складываем их во временное хранилище, которое опустошается раз в сутки после финальной сверки значений, что на большой дистанции позволит существенно экономить размер дампа базы данных.

Также следует упомянуть, что биржа Binance выгружает все сделки за день в 00:00 UTC. Сразу же после выгрузки на стороне сервиса запускается таска, которая аккумулирует все показатели торговавших пользователей, далее запрашивает данные у Binance и производит сверку, что все данные сходятся. Используя такую логику работы, избегаем складирования ненужных объемов данных, которые в дальнейшем никак не планируем использовать и, как следствие, уменьшается время на запросы к базе данных, так как отпадает необходимость обращаться в огромной базе. В случае, если какая-то часть данных не сходится, в отдельный микросервис приходит оповещение, о том, что у пользователя не сходятся итоговые цифры. Такие кейсы разбираем в частном порядке, но за основу и за легитимные данные для расчёта cashback’а используем данные с биржи Binance. Подобный кейс встречался единожды на тестовом аккаунте и был связан с ошибкой в таске при отладке.

3.3.3 Система конвертации и выплаты cashback

Для технической реализации механизма cashback’а, в момент регистрации пользователя, вместе с его основными кошельками, регистрируем 3 «невидимых» кошелька, которые используются для перевода денежных средств, через промежуточный хаб. С полной схемой перечисления средств cashback’а можно ознакомиться на рисунке 21.

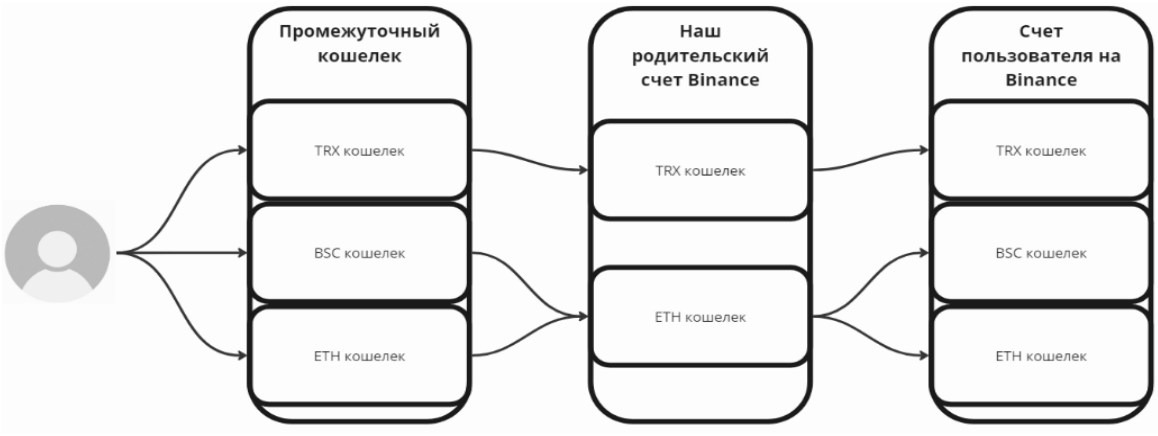


Рисунок 21 - Схема начисления денежных средств на счета пользователей

Полная схема выглядит следующим образом:

– каждую полученную транзакцию фиксируем в таблицу,

– запрашиваем кошелек куда нужно перевести средства пользователю,

– по данным события определяем каким методом нужно перевести средства: определяем через какую сеть будет перевод, вычисляем количество газа в данной сети, внутри сети это будет монетой этой сети (TRX, BNB, ETH соответственно), отправляем соответствующим методом,

– уведомляем сервис личного кабинета о статусе перевода. На данный момент статусы: «В процессе», «Успешно», «Ошибка».

С технической точки зрения, на Frontend и Backend виджет работает следующим образом: пользователь хочет вручную вывести денежные средства, он нажимает на виждет. Благодаря современным средствам JS у нас есть возможность отслеживать это событие, например, например, таким способом:

```
$(function() {
    document.getElementById(some_element).click();
});
```

Сразу же после нажатия отправляется GET запрос на Binance API, чтобы получить текущий курс монет по интересующим нас парам. Эти данные записываются во временную память и отображаются непосредственные

значения пользователю на frontend. Далее возможны два состояния: либо пользователь продолжает использовать виджет, либо прекращает взаимодействие (в качестве timeout используем время жизни вкладки браузера). Пользователь видит доступные к конвертации средства. Опять же, с помощью *document.getelementbyid.onchange(),* сравниваются введенное значение и доступное для вывода. В случае, если введено некорректное значение, логика не допускает пользователя до следующего шага. На следующем шаге пользователь нажимает кнопку «Конвертировать», а на сервер уходит POST запрос. На Backend дополнительно перепроверяем, доступно ли введенное количество. Если доступно – создаем таску на перевод средств, используя API Binance, если указанная сумма больше не валидна – оповещаем об этом пользователя и просим ввести сумму с учетом коррекции. Более наглядно со схемой можно ознакомиться на рисунке 22.

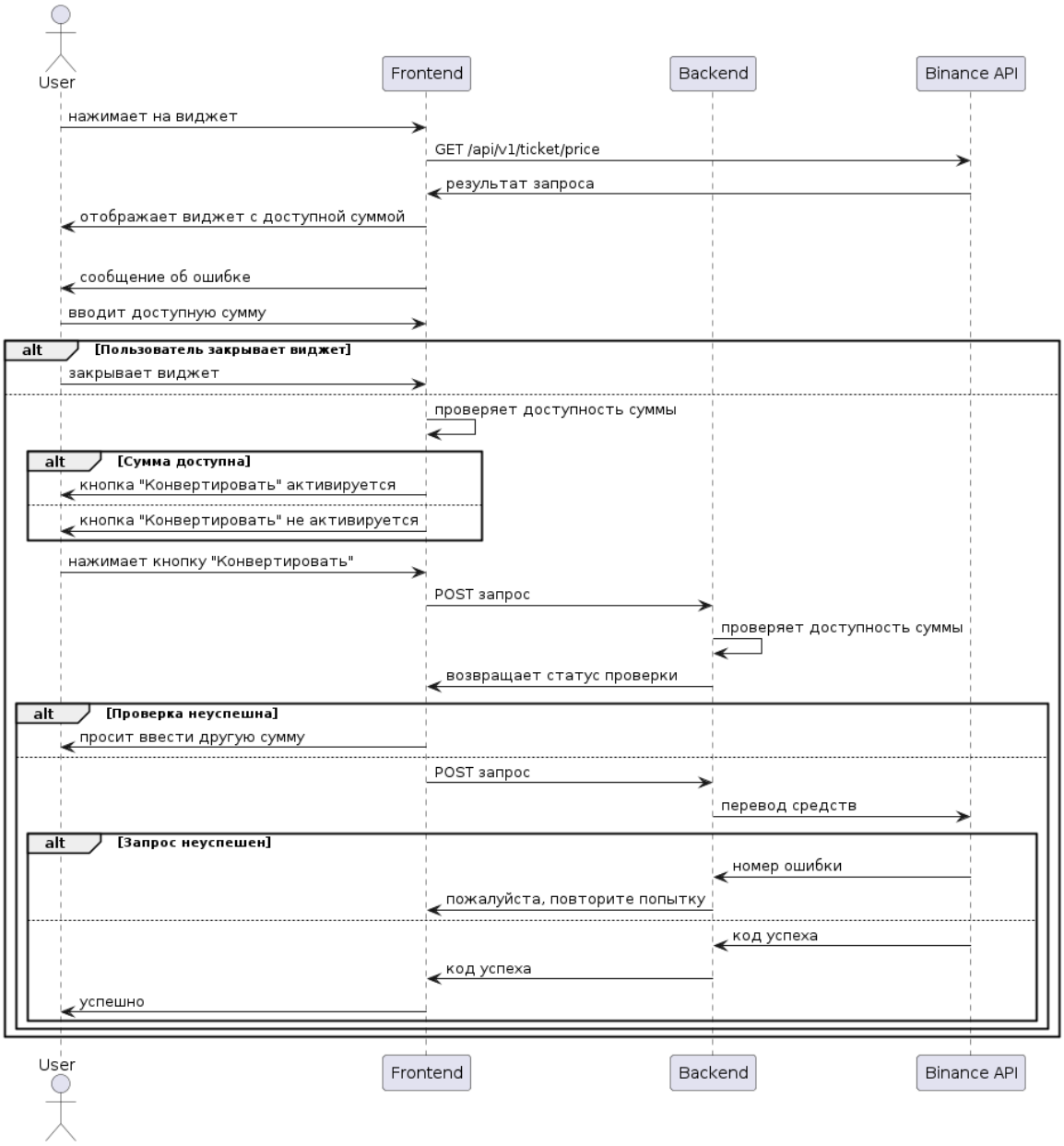


Рисунок 22 – UML диаграмма работы системы конвертации средств.

3.3.4 История начисления бонусов

Для реализации механизма фильтрации используется виджет whatchdog, который при открытии добавляет в ajax запрос новые заголовки в виде: *‘Start_date’, ‘End_date’, ‘Coin’, ‘field’*. Далее запрос уходит на backend, который возвращает уже отфильтрованный json, по указанным данным. На основании этого принудительно ререндерится страничку на frontend. В результате, нам удалось достичь эффекта SPA, в этой части, когда данные на

странице обновляются без перехода на другой URL или перезагрузки страницы, что положительно сказывается на быстродействии сервиса для пользователя.

Готовый интерфейс получившегося инструмента можно увидеть на рисунках 23 и 24.

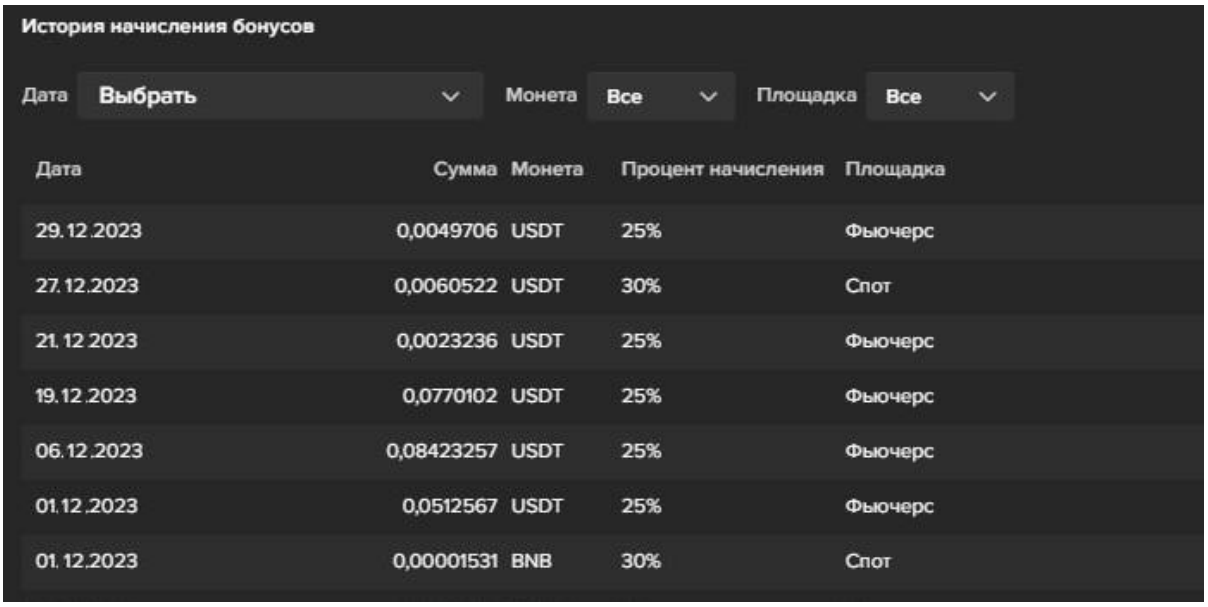


Рисунок 23 – История начисления бонусов

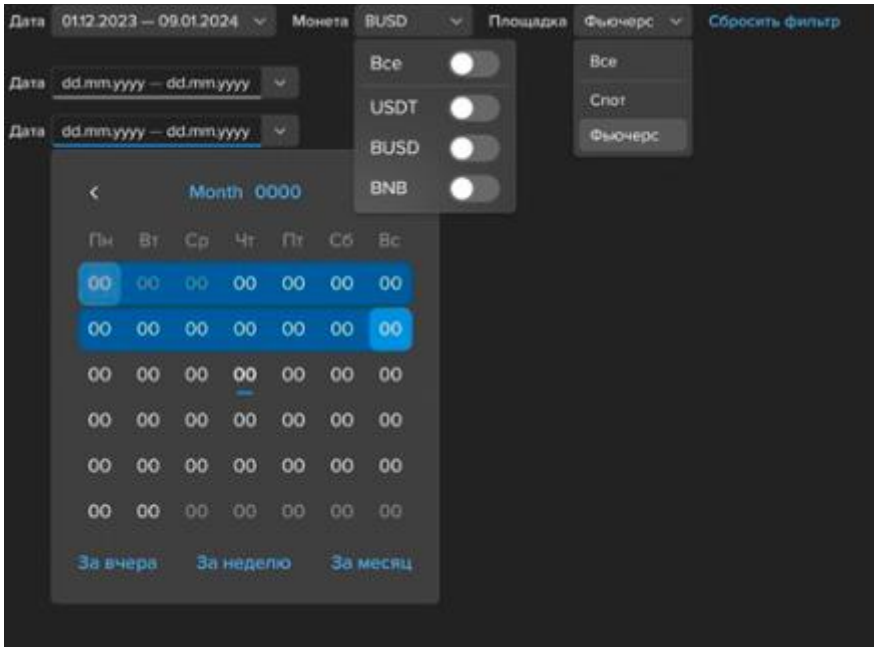


Рисунок 24 – Пример фильтра

3.4 Рефакторинг и тестирование

3.4.1 Рефакторинг

Из-за ограниченных сроков реализации в ходе реализации проекта были использованы разного рода неоптимальные пути реализации в угоду скорости разработки. В части функций упущены комментарии и аннотации типов. Последние два пункта, действительно, не влияют на производительность, но серьезно снижают время читаемости и, как следствие, поддерживаемость кода в будущем ввиду того, что команде требуется больше времени для погружения в суть той или иной функции, особенно если функция была написана в самом начале разработки.

«Костыль» – это не идеальное решение, чаще всего не оптимизированое с точки зрения эффективности и производительности [62]. В современном программировании применение «костылей» представляет собой практику внесения временных, неоптимальных или решений в код с целью обхода существующих проблем или ошибок. Однако, несмотря на возможность достижения временного успеха с использованием данной практики, она влечет за собой целый ряд серьезных проблем, подрывающих общую стабильность. Это приводит к тому, что обслуживание такого кода становится трудозатратным и подверженным ошибкам процессом. Такие временные решения, как правило, не являются тщательно протестированными и не обладают должной устойчивостью к различным сценариям использования. Это может привести к непредсказуемому поведению системы, внесению новых дефектов и в целом ухудшению качества программного продукта.

Таким образом, было принято решение, о необходимости в текущем режиме сначала покрыть весь код комментариями и аннотациями типов, а уже затем приступать к рефакторингу функций и классов. Такой подход позволил планомерно и последовательно покрыть все функции комментариями типа: «Общее описание функции, принимаемые аргументы и их описание с типом,

возврат с типом», при этом, когда время дошло до рефакторинга, нигде не запутаться и ничего не сломать.

Кроме того, было принято решение начать использовать систему статического анализа кода (далее – линтер). Система статического анализа кода – это инструмент для аудита кода, включающий в себя ряд анализаторов и расширяющий их функциональность. Она позволяет осуществлять запуск анализа для всего проекта или его части, а не отдельного файла, гибко настраивать игнорируемые ошибки и строки кода, непосредственно указывать используемые для проверки анализаторы, использовать конфигурационный файл, а также выводить информацию в различных представлениях [63]. В качестве линтера, на основании научно-исследовательских изысканий, было принято решение использовать pylint, как наиболее гибкий, быстрый и удобный в использовании из всех доступных open source инструментов.

После первичной настройки наиболее частыми недочетами были: broad-exception-caught, logging-fstring-interpolation, missing-function-docstring, а также line-too-long. С примером отчёта, полученного после запуска pylint можно ознакомиться на рисунке 25.

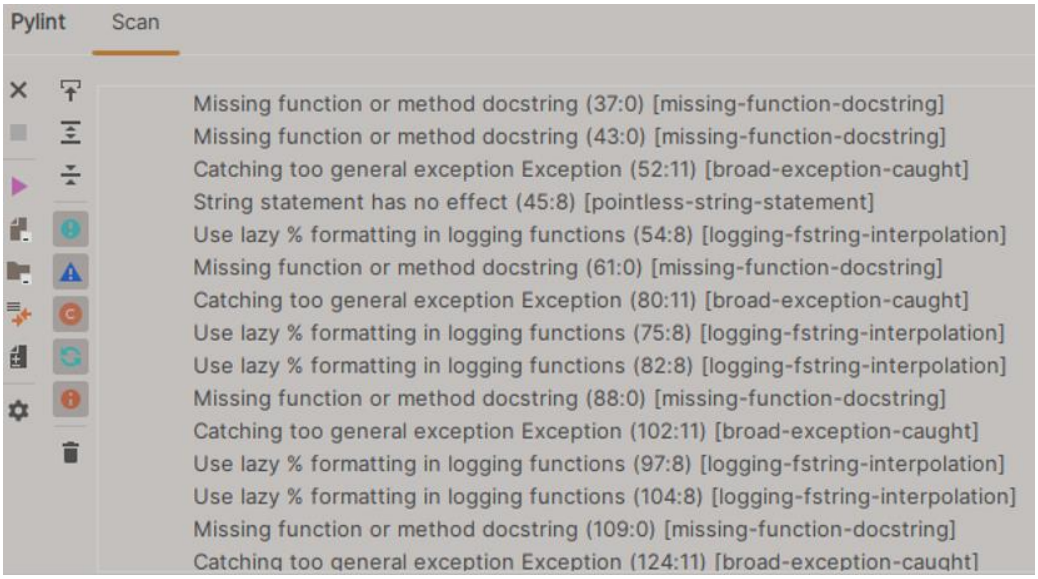


Рисунок 25 – Вывод анализа pylint.

Кроме общих недочётов, pylint подсвечивал (например, при ошибке too-many-return-statements – мы разбивали функцию со слишком большим количеством возможных возвратов на несколько отдельных, делая код более читаемым и простым для понимания) нам функции, которые нам предстояло отрефакторить, что и было успешно выполнено. В части функций было принято решение отказаться от рефакторинга, так как, их рефакторинг не добавил бы производительности системе.

3.4.2 Тестирование

Для оценки быстродействия разработанного сервиса был выбран метод нагрузочного тестирования [64]. Были сформированы тестовые запросы в количестве 100 штук на операцию, которые отправлялись асинхронно с разработанного web-сервиса и сервиса конкурента VatagaSPB. В качестве показателя измерения метрик считалось время от отправки команды до call-back ответа со стороны биржи.

Следует дать пояснения: url и тело запроса было получены с помощью утилиты Wireshark [65] на операционной системе Kali Linux [66]. После получения необходимых реквизитов были сформированы сначала тестовые пакеты данных, с помощью которых мы проверили, что сервер отвечает на такие запросы. Убедившись, что все работает, мы подготовили по каждой метрике пачки с запросами в количестве 100 (ста) штук и начали поочередно проводить тестирование. Тесты перепроверялись 3 раза, так что результаты можно считать достоверными. С результатами можно ознакомиться на рисунках 26-29.

```
Среднее время разработанного веб-сервера за 100 запросов-ответов 0.4709
VatagaSPB среднее время за 100 запросов-ответов : 0.5982
Средняя разница во времени: 0.1273
```

Рисунок 26 – Закрытие открытых позиций

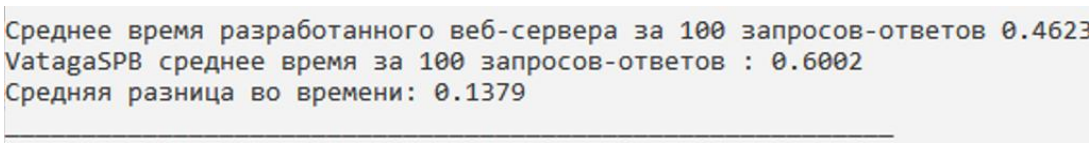


Рисунок 27 – Снятие лимитных заявок

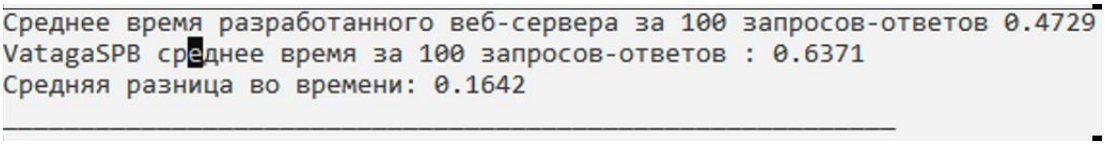


Рисунок 28 – Обновление данных по текущей цене позиции

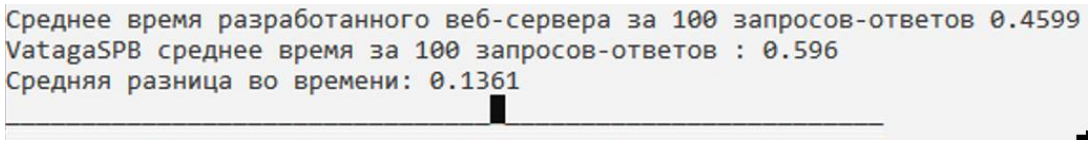


Рисунок 29 – Отправка данных между спотовым и фьючерсными счетами

Добиться полученных результатов уже на этапе MVP стало возможным за счет следующих факторов:

- разработанный web-сервис отправляет команды после получения action прямо на Binance, в то время как терминал отправляет данные сперва на сервер, после чего на биржу, что дает выигрыш в скорости;
- в web-сервисе используются и обрабатываются только релевантные данные, необходимые для конкретных операций, совершаемых трейдером, в то время как в терминале количество данных гораздо выше, как и на самой бирже Binance, именно поэтому данные обновляются быстрее;
- поскольку счета, открываемые пользователю, находятся внутри инфраструктуры Binance, топологически гораздо проще и быстрее оперировать с ними внутри биржи, контролируя это все запросами со стороны web-сервиса;
- при разборе продукта от конкурентов был обнаружен файл *.config в котором размещен XML-код с опциями. Как показали тесты, этот конфиг ощутимо влияет на быстродействие системы. Так, например, параметр

OrdersWith является поднастройкой потока, содержащего данные заявок и потока информации об изменении котировок в терминале. Параметр *RequestOrders* отвечает за право хранить определенное количество заявок локально. Каждый раз, когда пользователь нажимает кнопку "закрыть заявку" – программа обращается к своему словарю, где берет последнее значение, производит парсинг, находит нужное значение (все данные хранятся в неупорядоченном виде, так что возможности обратиться напрямую к элементу нет) и только после этого отправляет на сервер.

Разработанное web-приложение оптимизировано для максимальной скорости, минуя локальный поиск заявок и используя централизованные ресурсы. Это позволяет нам выиграть драгоценные миллисекунды за счет уменьшения задержек, характерных для локальных операций. Функционал продукта разработан с учетом требований безопасности и оптимизирован для быстродействия, что идеально подходит для динамичной торговой среды.

Предполагается, что разработанное web-приложение способно обеспечить не только быстродействие всей системы, но и выдержать существенную нагрузку пользователей, работающих одновременно. С целью проверки устойчивости системы к нагрузкам, был развернут Locust. Это фреймворк тестирования, написанный на Python. Locust предоставляет возможность эмулировать запросы большого числа пользователей одновременно по определенным URL адресам в случайно порядке, что максимально приближает такое тестирование к реальным условиям эксплуатации. Для тестов были написаны запросы по всем основным АПИ эндпоинтам, к которым обращаются пользователи в ходе взаимодействия с web-приложением. С графиками, построенными по результатам тестирования можно ознакомиться на рисунке во вложении Е.

Исходя из информации, полученной в ходе тестов, можно сделать вывод, что при текущей конфигурации сервера (12th Gen Intel Core i5-12400F 2,4 ГГц, 6 ядер, 16 ГБ DDR4, 480 ГБ SSD, скорость соединения 500 Мбит/с) сервис без проблем может выдерживать 10 тысяч пользователей при 500 запросах в

секунду в пике, далее наблюдается существенное падение скорости выполненных запросов в секунду. При этом важно отметить, что в этом случае обращения на сервер шли непрерывно на разные URL адреса, включая GET, POST, PUT запросы, взаимодействующие со всеми развернутыми сервисами. С учетом общего количества целевой аудитории, для которой разрабатывалось web-приложение, на данный момент указанная конфигурация успешно справляется с количеством запросов. Если открыть мониторинг ресурсов встроенной в Docker командой «docker stats --all», можно следить за показателями каждого из контейнеров по таким параметрам как: загруженность центрального процессора в процентах, загруженность памяти, линии интернет и прочими. При тестировании выявлено, что Gunicorn перестает справляться с загруженностью, начинает отклоняет запросы, становится так называемым «бутылочным горлышком». Это значит, что часть кода до него выполняется успешно, а уже за ним начинаются проблемы с производительностью. Критических ошибок как таковых не происходит, так как весь код покрыт обработкой исключений.

Решить проблему и повысить количество обрабатываемых запросов в секунду можно несколькими путями:

- увеличение мощностей серверного оборудования;
- горизонтальное масштабирование:
    - балансировка нагрузки с помощью NGINX поможет разделять запросы между несколькими инстансами Gunicorn;
    - кэшировать большее количество данных, так как кэш работает быстрее за счёт особенностей NoSQL баз данных;
    - репликация базы данных PostgreSQL:
        - master-slave позволит распределить чтение данных между несколькими сервера с PostgreSQL;
        - шейдеринг – это разбитие базы данных на части (шейдеры) с последующим размещением на разные сервера;

- имплементация распределенной очереди задач на поступающие запросы, разделение между несколькими воркерами;

- кластеризация.

Таким образом, можно сделать вывод, что архитектура разработанного web-приложение пригодна для масштабирования.

Также были проведены тесты на наличие уязвимостей в сервисе. Поиск уязвимостей был выполнен с помощью инструмента ZAP (Zed Attack Proxy) [67]. Этот инструмент в автоматическом режиме позволяет обнаруживать такие уязвимости как: SQL-инъекции, межсайтовый скриптинг и другие. После выполнения сканирования генерируется отчёт по одному из шаблонов на выбор, в котором описываются найденные уязвимости, а также рекомендации по их устранению. Важно отметить, что ZAP в автоматическом режиме сканирует абсолютно все дочерние страницы в указанном URL. С итоговым отчётом можно ознакомиться в приложении Е.

Из отчёта следует, что в разработанном web-приложении найдено 6 уязвимостей с уровнем риска «низкий». Проанализировав каждую из них, можно сделать вывод, что прямой угрозы ни одна не несет. Все из них, включая информационные, несут скорее рекомендательный характер, например: «Раскрытие отметки времени – Unix» [68] или «Раскрытие информации – подозрительные комментарии». Кроме того, был проведен анализ безопасности web-приложения от компании ООО «Ватага», с которым можно ознакомиться в приложении Г. Разработанному web-приложению удалось достичь лучших показателей безопасности в сравнении с web-приложением конкурентов.

Полученная в ходе тестирования информация об уязвимостях позволяет сделать вывод, что web-приложение отвечает всем требованиям безопасности и не содержит критических уязвимостей.

ЗАКЛЮЧЕНИЕ

В результате проведенного исследования, посвященного разработке web-приложения для торговли на бирже Binance, удалось успешно достичь

поставленных целей и решить поставленные задачи. В рамках работы была проведена комплексная аналитическая работа, включающая в себя изучение предметной области, анализ существующих аналогов, определение методологии разработки, а также выбор подходящего технологического стека и архитектурного решения.

Анализ предметной области и существующих аналогов позволил определить ключевые требования к разрабатываемому web-приложению. Было установлено, что на рынке присутствует ограниченное количество специализированных решений, предоставляющих удобный и быстрый доступ к торговле криптовалютой внутридневными трейдерами. Анализ существующих аналогов, таких как web-приложение от ООО «Ватага», выявил ряд недостатков, связанных с медленной обработкой запросов, ограниченным функционалом и неудобным интерфейсом.

Выбор технологического стека и архитектурного решения был основан на анализе современных трендов в веб-разработке, а также требованиях к быстродействию и масштабируемости разрабатываемого сервиса. Было решено использовать VueJS в качестве фреймворка для frontend разработки, Python с фреймворком Django и Django Rest Framework для backend разработки, PostgreSQL в качестве базы данных, Redis для кеширования, Celery для асинхронной обработки задач, Celery-beat для планирования задач, websockets для двусторонней коммуникации в реальном времени, а также Docker для контейнеризации приложения и NGINX для обработки HTTP-запросов.

Процесс разработки включал в себя следующие этапы: проектирование, разработка, тестирование и отладка. В процессе разработки были использованы различные библиотеки и фреймворки, которые позволили создать функциональное и удобное web-приложение, обеспечивающее быстрый доступ к торговым инструментам биржи Binance.

Апробация разработанного приложения на реальных пользователях показала высокую эффективность и удобство использования. Были получены

положительные отзывы от пользователей, отмечавших быстродействие приложения, удобный интерфейс и широкий функционал.

Сравнительный анализ разработанного приложения с аналогом от ООО «Ватага» продемонстрировал значительное улучшение скорости ответа на запросы, связанные с закрытием позиций, снятием лимитных заявок и обновлением данных о текущей цене. Это достигнуто за счет использования более эффективного технологического стека и оптимизации архитектуры.

В рамках разработки приложения были реализованы следующие ключевые функции:

– быстрый доступ к торговым инструментам биржи Binance: разработан интуитивно понятный интерфейс для торговли криптовалютами, обеспечивающий быстрый доступ к нужным инструментам;

– режим реального времени (real-time) с использованием websockets: приложение обеспечивает пользователя актуальной информацией о текущих ценах и динамике рынка, что позволяет принимать быстрые и обдуманные решения;

– инструменты для технического анализа: в приложении реализованы базовые инструменты для технического анализа графиков, что позволяет пользователям проводить анализ динамики цен и принимать более информированные решения;

– безопасность данных: приложение обеспечивает безопасность данных пользователей с использованием современных механизмов шифрования и аутентификации.

Результаты исследования позволяют сделать следующие выводы:

– разработанное web-приложение для торговли на бирже Binance является эффективным и удобным инструментом для внутридневных трейдеров;

– использование современного технологического стека и оптимизация архитектуры позволили добиться значительного увеличения скорости ответа приложения;

– разработанное приложение предоставляет пользователям удобный и функциональный интерфейс, что позволяет проводить торговые операции более эффективно и удобно.

Исследование также выявило ряд перспектив развития web-приложения:

– добавление новых инструментов для торговли,

– интеграция с другими биржами криптовалют,

– разработка полноценного мобильного приложения.

В целом, разработанное web-приложение для торговли на бирже Binance является важным шагом в развитии инструментов для внутридневных трейдеров и обеспечивает им более эффективный и удобный доступ к рынку криптовалют.

УТВЕРЖДАЮ
HR-Директор Зайцев А.В.
ООО «Вайтлист»
«___»_____ 2024 г.

СПРАВКА (АКТ ВНЕДРЕНИЯ)

О внедрении результатов выпускной квалификационной работы студента магистратуры ФГАОУ ВО «УрФУ имени первого Президента России Б. Н. Ельцина» Новикова Романа Игоревича.

Результаты выпускной квалификационной работы Новикова Романа Игоревича на тему Разработка клиент-серверной части приложения для торговли на бирже Binance» одобрены, как имеющие практическую значимость и приняты к использованию ООО «Вайтлист».

Разработанное автором выпускной квалификационной работы приложение используется в работе ООО «Вайтлист» на повседневной основе и обеспечивает функционирование части внутренних продуктов, позволяющих осуществлять торговлю на бирже Binance.

_____ 2024 г.

HR-директор Зайцев А.В.

м.п.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. «Дейтрейдинг (Day Trading): основы стратегии торговли внутри дня в 2024» [Электронный ресурс]. URL: https://investonomic.ru/deytreyding-day-trading-osnovy-strategii-torgovli-vnutri-dnya/. (Дата обращения 01.02.2024).

2. «How the Top Cryptocurrencies Performed in 2021» [Электронный ресурс]. URL: https://www.visualcapitalist.com/how-the-top-cryptocurrencies-performed-in-2021/. (Дата обращения 01.02.2024).

3. «Intraday или внутридневная торговля акциями: что это и как работает?» [Электронный ресурс] URL: https://streetinvestor.ru/vnutridnevnoj-trejding-podrobnyj-gajd/ (дата обращения 22.02.2024).

4. «Что такое кредитное плечо: полный гайд для начинающих» [Электронный ресурс] URL: https://www.litefinance.org/ru/blog/for-beginners/kreditnoe-plecho/ (дата обращения 01.02.2024).

5. «Стоп-лосс и тейк-профит. Как ограничить убытки и при чем тут лось?» [Электронный ресурс] URL: https://quote.rbc.ru/news/training/5e1dffb59a79476d9266ff63 (дата обращения 22.02.2023).

6. «Что такое централизованная биржа (CEX)?» [Электронный ресурс] URL: https://www.okx.com/ru/learn/what-is-centralized-exchange (дата обращения 24.05.2023).

7. «How to Choose a Centralized Cryptocurrency Exchange (CEX)» [Электронный ресурс] URL: https://www.coingecko.com/learn/what-are-centralized-crypto-exchanges-cex (дата обращения 15.12.2023).

8. «Что такое децентрализованная биржа (DEX) и как это работает [Электронный ресурс] URL: https://ru.beincrypto.com/learn/chto-takoe-deczentralizovannye-birzhi-i-zachem-oni-vam/ (дата обращения 02.02.2024).

9. «Что будет с регулированием криптовалют в России в 2024 году» [Электронный ресурс]. URL: https://www.rbc.ru/crypto/news/6597fe339a79478ed72b38dd. (Дата обращения 01.02.2024).

10. «SWOT-анализ с примерами» [Электронный ресурс]. URL: https://esputnik.com/blog/swot-analiz-s-primerami. (Дата обращения 05.03.2024).

11. Манин Е. И. «Проектирование и разработка web-приложения для торговли на платформе Binance с использованием продуктового подхода», раздел 1.3 «Анализ рынка РФ, конкурентных решений».

12. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Седьмое издание / [пер. с англ.] – М.: Издательство «Олимп-Бизнес», 2021. – 792 с

13. «Waterfall Methodology: A Comprehensive Guide» [Электронный ресурс]. URL: https://www.atlassian.com/agile/project-management/waterfall-methodology. (Дата обращения 23.02.2024).

14. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Седьмое издание / [пер. с англ.] – М.: Издательство «Олимп-Бизнес», 2021. – 38 с

15. Кагазжев А.О., Криштоп В.В., Гоплачев А.С., Белова А.А. "Преимущества и недостатки Agile-методологий менеджмента". Журнал "Экономика и предпринимательство" №11 (112) 2019. С. 634-637.

16. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Седьмое издание / [пер. с англ.] – М.: Издательство «Олимп-Бизнес», 2021. – 252 с

17. «Scrum: что это и зачем нужно» [Электронный ресурс]. URL: https://scrumtrek.ru/blog/agile-scrum/3777/scrum-chto-eto/. (Дата обращения 22.03.2024).

18. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Седьмое издание / [пер. с англ.] – М.: Издательство «Олимп-Бизнес», 2021. – 244 с

19. Манин Е. И. «Проектирование и разработка web-приложения для торговли на платформе Binance с использованием продуктового подхода», разделы 1.3 и 1.4.

20. Тарнавская, О. А. Основные этапы проектирования информационной системы / О. А. Тарнавская // Информационные технологии и автоматизация управления : Материалы XIV Всероссийской научно-практической конференции студентов, аспирантов, работников образования и промышленности, Омск, 26–27 мая 2023 года / Отв. редактор А.В. Никонов. – Омск: Омский государственный технический университет, 2023. – С. 220-224. – EDN ZMHWEQ.

21. «Clean Architecture» [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/732178/. (Дата обращения 24.03.2024).

22. «Accelerate how you build, share, and run applications» [Электронный ресурс]. URL: https://www.docker.com/. (Дата обращения 12.02.2024).

23. Гайнанова, Р. Ш. «Создание клиент-серверных приложений» / Р. Ш. Гайнанова, О. А. Широкова // Вестник Технологического университета. – 2017. – Т. 20, № 9. – С. 79-84. – EDN YNTPOJ.

24. «Паттерны для новичков: MVC vs MVP vs MVVM» [Электронный ресурс]. URL: https://habr.com/ru/articles/215605/. (Дата обращения 12.02.2024).

25. «Простая архитектура с использованием MVVM и делегатов в Android. Оптимальное решение для малых проектов» [Электронный ресурс]. URL: https://habr.com/ru/articles/776344/. (Дата обращения 12.02.2024).

26. «Введение в Django» [Электронный ресурс]. URL: https://metanit.com/python/django/1.1.php (Дата обращения 14.02.2024).

27. «Event-driven architecture» [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Event-driven_architecture (Дата обращения 15.02.2024).

28. «Docker docs» [Электронный ресурс]. URL: https://docs.docker.com/ (Дата обращения 15.02.2024).

29. «Docker Monitoring Tools for Docker Container» [Электронный ресурс]. URL: https://www.solarwinds.com/server-application-monitor/use-cases/docker-monitoring (Дата обращения 19.02.2024).

30. «What is Prometheus?» [Электронный ресурс]. Режим доступа: https://prometheus.io/docs/introduction/overview/ (Дата обращения 19.02.2024).

31. «Что такое Server-Side Rendering» [Электронный ресурс]. URL: https://academy.yandex.ru/journal/server-side-rendering (Дата обращения: 19.02.2024)

32. «Client-side Rendering (CSR)» [Электронный ресурс]. URL: https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering (Дата обращения: 19.02.2024)

33. «SPA (Single-page application)» [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (Дата обращения: 19.02.2024)

34. «SINGLE-PAGE APPLICATION VS MULTI-PAGE APPLICATION: PROS, CONS, AND WHICH IS BETTER?» [Электронный ресурс]. URL: https://lvivity.com/single-page-app-vs-multi-page-app#:~:text=A%20Multi-page%20Application%20is%20a%20web%20application%20consisting,to%20a%20new%20page%20displayed%20in%20the%20browser (Дата обращения: 19.02.2024)

35. «nginx» [Электронный ресурс]. URL: https://nginx.org/ru/ (Дата обращения: 23.02.2024)

36. «HTTP server project» [Электронный ресурс]. URL: https://httpd.apache.org/ (Дата обращения: 11.03.2024)

37. «Рейтинг веб-серверов в зоне ru» [Электронный ресурс]. URL: https://itrack.ru/research/web-server/ (Дата обращения: 19.09.2023)

38. «Python» [Электронный ресурс]. URL: https://www.python.org/ (Дата обращения: 19.02.2024)

39. «Django documentation» [Электронный ресурс]. URL: https://docs.djangoproject.com/en/5.0/ (Дата обращения: 11.03.2024)

40. «Flask» [Электронный ресурс]. URL: https://flask.palletsprojects.com/en/3.0.x/ (Дата обращения: 11.03.2024)

41. «AIOHTTP» [Электронный ресурс]. URL: https://docs.aiohttp.org/en/stable/index.html (Дата обращения: 17.05.2024)

42. «Django REST framework» [Электронный ресурс]. URL: https://www.django-rest-framework.org/ (Дата обращения: 17.05.2024)

43. «Celery - Distributed Task Queue» [Электронный ресурс]. URL https://docs.celeryq.dev/en/stable/index.html (Дата обращения: 17.05.2024)

44. «Celery - распределенная очередь задач» [Электронный ресурс]. URL: https://django.fun/docs/celery/5.1/ (Дата обращения: 17.05.2024)

45. «Redis» [Электронный ресурс]. URL: https://redis.io/ (Дата обращения: 17.05.2024)

46. «RabbitMQ 3.13» [Электронный ресурс]. URL: https://www.rabbitmq.com/docs (Дата обращения: 11.05.2024)

47. «django-celery-beat» [Электронный ресурс]. URL: https://github.com/celery/django-celery-beat (Дата обращения: 11.05.2024)

48. «PostgreSQL 16.3 Documentation» [Электронный ресурс]. URL: https://www.postgresql.org/docs/current/ (Дата обращения: 11.05.2024)

49. «MySQL» [Электронный ресурс]. URL: https://dev.mysql.com/doc/ (Дата обращения: 11.05.2024)

50. «What if … MySQL’s Repeatable Reads Cause You to Lose Money?» [Электронный ресурс]. URL: https://www.percona.com/blog/what-if-mysqls-repeatable-reads-cause-you-to-lose-money/ (Дата обращения: 13.05.2024)

51. «10 Best JavaScript Frameworks in 2024» [Электронный ресурс]. URL: https://hackr.io/blog/best-javascript-frameworks (Дата обращения: 17.05.2024)

52. «Binance API docs» [Электронный ресурс]. URL: https://binance-docs.github.io/apidocs/spot/en/#change-log (Дата обращения: 17.05.2024)

53. «WebSockets: что это, как работает, плюсы и минусы» [Электронный ресурс]. URL: https://media.mts.ru/business/206250-chto-takoe-websockets/ (Дата обращения: 17.05.2024)

54. «Отличия HTTP и Websockets» [Электронный ресурс]. URL: https://nachniznanie.ru/otliciya-http-i-websockets/ (Дата обращения: 27.02.2024)

55. «Обзор протокола HTTP» [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview?ref=dtf.ru (Дата обращения: 27.02.2024)

56. «WebSocket: что это, когда следует использовать и какие преимущества дает» [Электронный ресурс]. URL: https://cloud.vk.com/blog/websocket-kogda-sleduet-ispolzovat-i-preimushhestva (Дата обращения: 27.04.2024)

57. «Использование принципов асинхронного программирования при разработке веб-приложений» [Электронный ресурс]. URL: https://elibrary.ru/item.asp?id=44441883. (Дата обращения 27.04.2024).

58. Агапов Ю.Г., Подвесовская М.А. «Сравнительный анализ библиотек языка Python для изучения методов визуализации данных». Журнал "Информационные технологии. Проблемы и решения" №2 (19) 2022. С. 20-26.

59. Колесников А.О. «Идентификация пользователей клиент-серверных приложений с помощью Jwt-токена». Сборник статей XXXVI международной научно-практической конференции. Москва, 2021. С. 42-43.

60. Старанцова Е.В. «Методы хеширования паролей». Сборник научных трудов "Пути инновационного развития науки и образования в современных условиях". Миасс, 2023. С. 387-394.

61. Eum S., Kim H., Song M., Seo H. «Optimized Implementation of Argon2 Utilizing the Graphics Processing Unit». «Applied Sciences» (Switzerland). 2023. Т. 13. № 16. С. 9295.

62. «Как правильно использовать костыли в разработке и не наплодить ошибок» [Электронный ресурс]. URL:

https://skillbox.ru/media/code/kak_pravilno_ispolzovat_kostyli_v_razrabotke/. (Дата обращения 27.04.2024).

63. Gimatdinov D.M., Gerasimov A.Y., Privalov P.A., Butkevich V.N., Chernova N.A., Gorelova A.A. «Automated Framework for Testing Source Code Static Analysis Tools». «Proceedings of the Institute for System Programming of the RAS». 2021. Т. 33. № 3. С. 41-50.

64. «Нагрузочное тестирование: что это, виды и этапы» [Электронный ресурс]. URL: https://school.kontur.ru/publications/2670 (Дата обращения: 24.04.2024)

65. «Wireshark» [Электронный ресурс]. URL: https://www.wireshark.org/ (Дата обращения: 24.04.2024)

66. «Kali docs» [Электронный ресурс]. URL: https://www.kali.org/docs/ (Дата обращения: 24.04.2024)

67. «ZAP» [Электронный ресурс]. URL: https://www.zaproxy.org/ (Дата обращения: 21.05.2024)

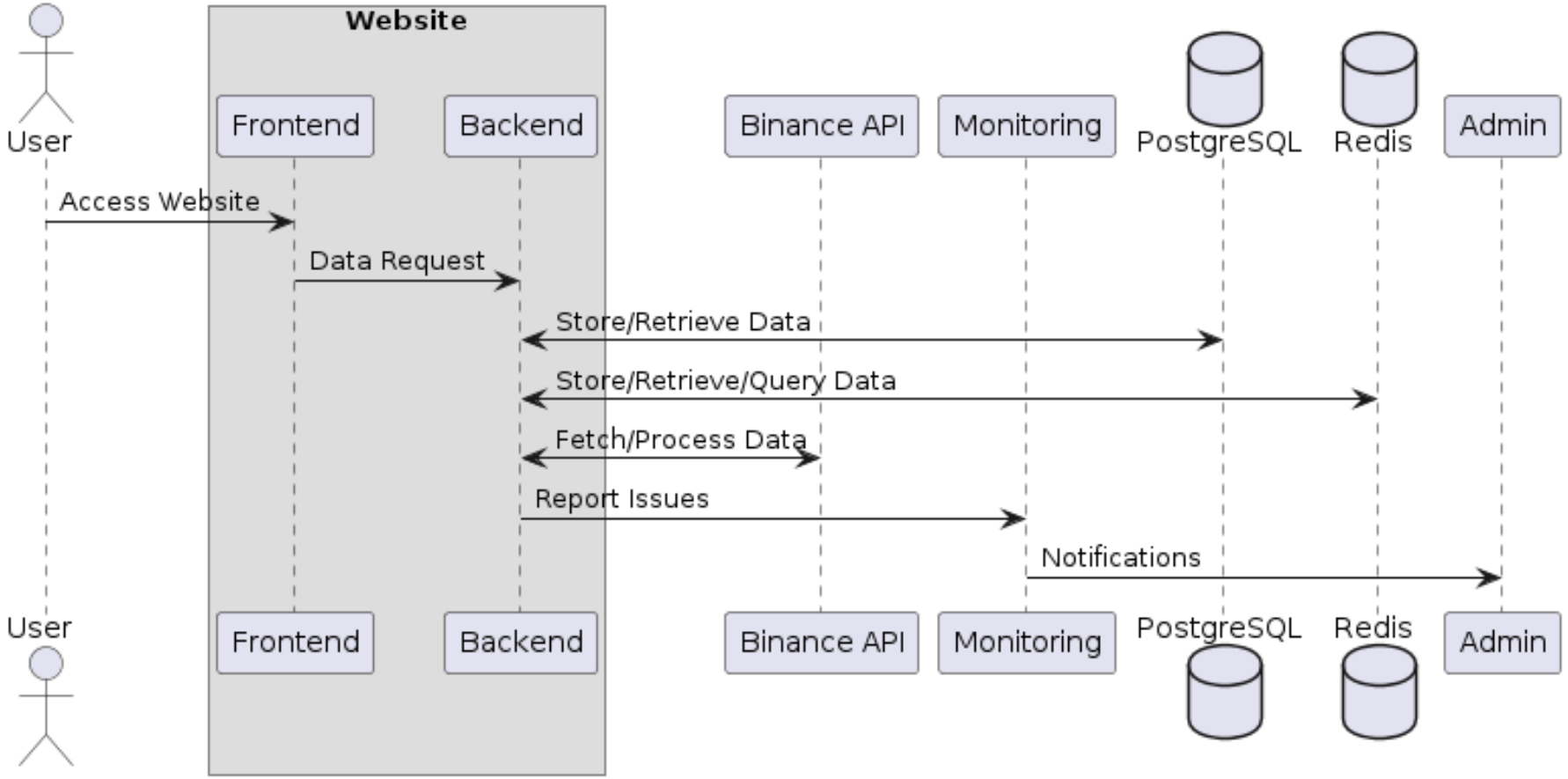
68. «CWE-200: Exposure of Sensitive Information to an Unauthorized Actor» [Электронный ресурс]. URL: https://cwe.mitre.org/data/definitions/200.html (Дата обращения: 21.05.2024)

ПРИЛОЖЕНИЕ А – СРАВНЕНИЕ AGILE И WATERFALL ПОДХОДОВ.

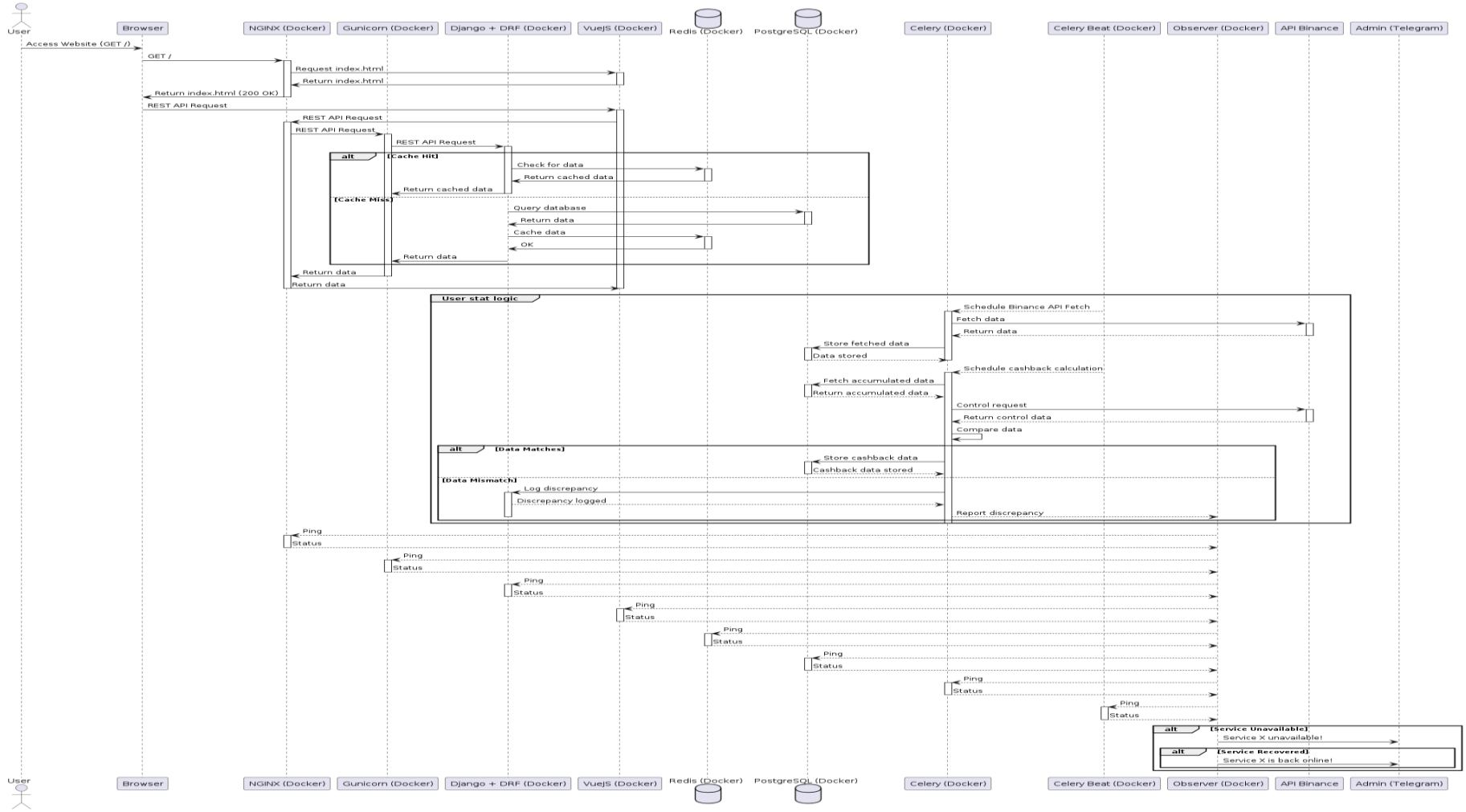
Характеристика	**Agile**	**Waterfall**
Философия	Гибкий подход, акцент на изменениях	Строгий последовательный процесс
Требования	Могут изменяться по ходу процесса ведения проекта	Фиксируются на начальных этапах, изменениям подвергаются редко
Итерации	Итеративная разработка	Итерации отсутствуют как таковые
Управление рисками	Осуществляется через регулярный сбор обратной связи и быстрые корректировки курса	Осуществляется преимущественно через предиктивную модель на ранних этапах ведения проекта. Трудно управлять на поздних стадиях
Прозрачность	Высокая из-за регулярных встреч, обсуждений и демонстрации продукта	Недостаточная из-за отсутствия регулярных встреч
Участие стэйкхлдеров	Активно участвуют в процессе разработки	Участие ограничено, так как обычно есть изначальные требования ТЗ, которым необходимо следовать

Характеристика	**Agile**	**Waterfall**
Гибкость планирования	Высокая, регулярное планирование на короткий срок	Низкая, большая часть планирования осуществляется на начальной стадии
Отказоустойчивость продукта	Высокая из-за регулярного тестирования продукта по итогам каждой итерации	Низкая и ресурсоемкая на исправления из-за проведения тестирования либо на завершающей стадии всего проекта, либо на этапе окончания очередной вехи
Процесс разработки	Итеративный и инкрементальный	Линейный и последовательный
Применимость	В проектах, где требуются быстрые изменения и адаптация к изменениям	В проектах со стабильной средой и с четко определенными требованиями

ПРИЛОЖЕНИЕ Б – ОБЩАЯ СХЕМА ПРИЛОЖЕНИЯ, ВЗАИМОДЕЙСТВИЕ ТЕХНОЛОГИЙ



ПРИЛОЖЕНИЕ В – UML ДИАГРАММА ЛОГИКИ РАБОТЫ WEB-ПРИЛОЖЕНИЯ

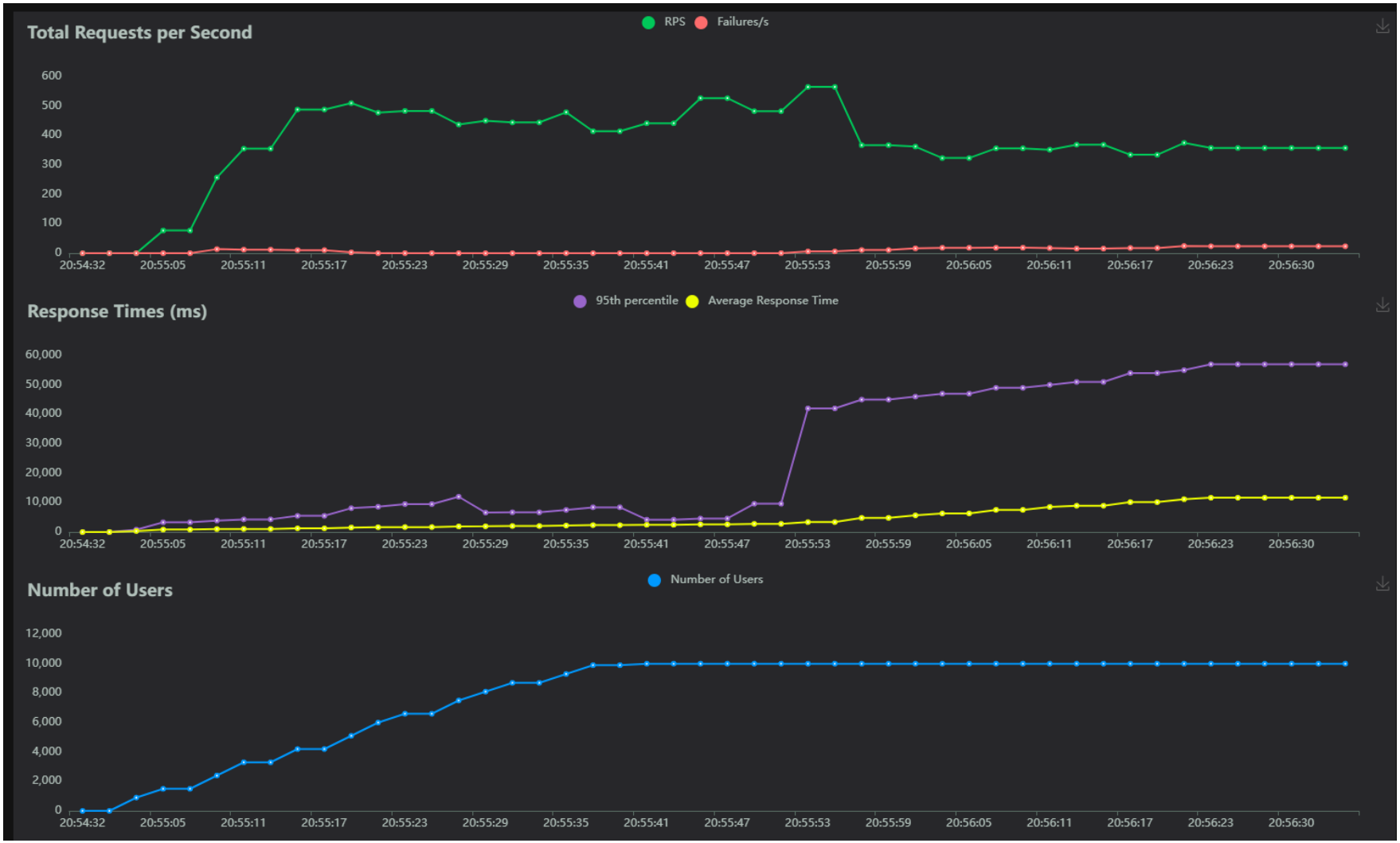


ПРИЛОЖЕНИЕ Г – ТАБЛИЦА СРАВНЕНИЯ ПОПУЛЯРНЫХ PYTHON BACKEND ФРЕЙМВОРКОВ

Характеристика	**Django**	**Flask**	**AioHTTP**
Тип фреймворка	Полноценный MVC веб-фреймворк	Микрофреймворк для веб-приложений	Асинхронный веб-фреймворк
ORM (Object-Relational Mapping)	Встроенный ORM	Поддержка сторонних ORM (например, SQLAlchemy)	Отсутствует
Шаблонизация	Встроенная система шаблонов	Поддержка шаблонизации через Jinja2	Отсутствует
Аутентификация и авторизация	Встроенные средства аутентификации и авторизации	Поддержка сторонних библиотек аутентификации и авторизации	Отсутствует
Административный интерфейс	Встроенный административный интерфейс	Отсутствует	Отсутствует
Гибкость и расширяемость	Высокая	Высокая	Высокая
Популярность	Очень популярен	Популярен, но менее, чем Django	Популярен в сообществе асинхронного Python

ПРИЛОЖЕНИЕ Д – ТАБЛИЦА СРАВНЕНИЯ MYSQL И POSTGRESQL

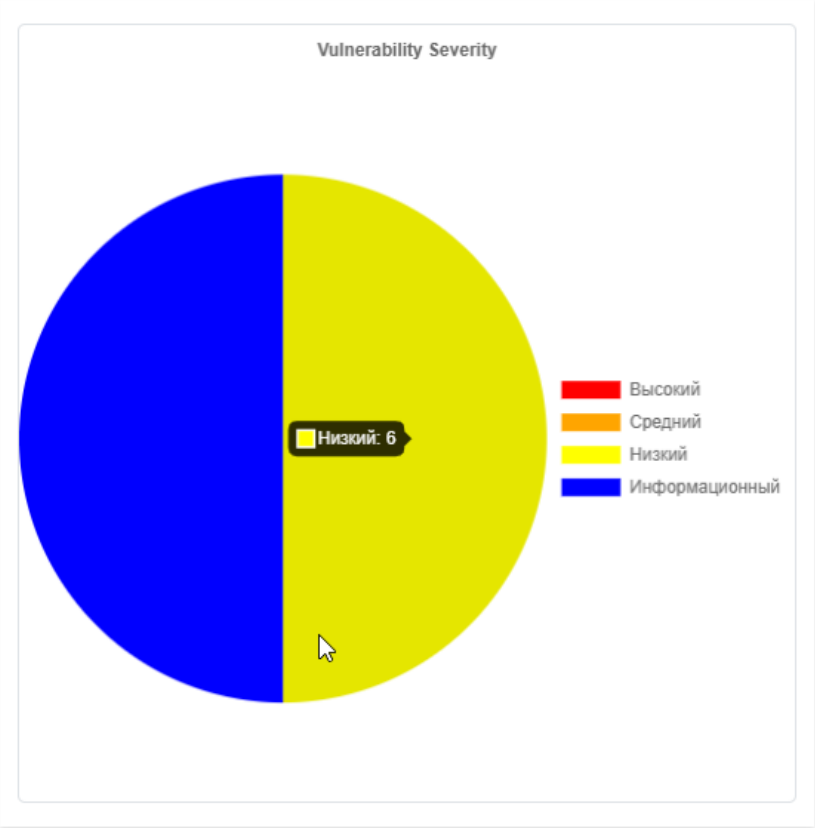
Характеристика	**MySQL**	**PostgreSQL**
Типы данных	Ограниченный набор, ориентация на простые приложения	Более широкий спектр, включая массивы, hstore (key-value), бинарные JSON и прочие
Индексирование	Только B-tree	B-tree, Generalized Search Tree (обобщенное поисковое дерево), Generalized Inverted Index
Репликация	Только Мастер-слейв	Мильти-мастер
Транзакции	Используется многоверсионный контроль параллелизма	Поддержка изолированности транзакций, атомарные транзакции и точки сохранения
Производительность	Высокая	Средняя
Масштабируемость	+	+
Стоимость	Есть платная и бесплатная версии	Бесплатна


Total Requests per Second
RPS
Failures/s
600
500
400
300
200
100
0
20:54:32
20:55:05
20:55:11
20:55:17
20:55:23
20:55:29
20:55:35
20:55:41
20:55:47
20:55:53
20:55:59
20:56:05
20:56:11
20:56:17
20:56:23
20:56:30
Response Times (ms)
95th percentile
Average Response Time
60,000
50,000
40,000
30,000
20,000
10,000
0
Number of Users
12,000
10,000
8,000
6,000
4,000
2,000
0

Отчет о сканировании ZAP

Создано чт, 06 мая 2024 02:05:40
ZAP Version: 2.15.0
ZAP is supported by the Crash Override Open Source Fellowship

Most Severe Alert
Low



ПРИЛОЖЕНИЕ 3 – ОТЧЕТ О БЕЗОПАСНОСТИ WEB-ПРИЛОЖЕНИЯ КОНКУРЕНТА ООО «ВАТАГА»

Отчет о сканировании ZAP

Создано чт, 06 мая 2024 03:37:23
ZAP Version: 2.15.0
ZAP is supported by the Crash Override Open Source Fellowship

Most Severe Alert
Medium