

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**«МИНИ-ПРИЛОЖЕНИЕ В TELEGRAM ДЛЯ
МОНИТОРИНГА И АНАЛИЗА
КРИПТОВАЛЮТНЫХ АКТИВОВ»**

Студент Пивкин Данила Александрович (подпись, дата) 11.06.2025 (ФИО полностью)

Руководитель Гурин Е.И. (подпись, дата) 11.06.2025 (фамилия, инициалы)

Нормоконтролёр Бычков А.С. (подпись, дата) 11.06.2025 (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой М.А. Митрохин (подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
16 июня 2025 г. № 37)

Секретарь ГЭК Е.И. Гурин (подпись) (фамилия, инициалы)
Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» май 2025 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Пивкин Данила Александрович гр. 21ВВП1 факультета ФВТ

2. Тема работы «Мини-приложение в Telegram для мониторинга и анализа криптовалютных активов»

Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.

3. Руководитель бакалаврской работы Гурин Е.И.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать мини-приложение в Telegram для мониторинга и анализа криптовалютных активов. Реализовать архитектуру клиент-серверного взаимодействия. Язык программирования – Python и JavaScript. Клиентская часть должна быть реализована с использованием фреймворка Vue.js 3, серверная – на FastAPI. В качестве системы управления базами данных использовать PostgreSQL. Приложение должно поддерживать аутентификацию через Telegram WebApp, обработку данных о курсах криптовалют, управление пользовательскими уведомлениями с настраиваемой периодичностью (минута/час/день), а также интеграцию с CoinGecko API для получения актуальных котировок. Backend-часть должна включать фоновые задачи для регулярного обновления данных через APScheduler.

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметных областей: изучить существующие решения для мониторинга криптовалют, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, изучить вопрос работы с API криптобирж. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал Visual Studio Code для разработки программы мини-приложения в Telegram.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	20	06.05.2025 – 08.05.2025	<i>Е</i>
2	Анализ требований	24	09.05.2025 – 10.05.2025	<i>Е</i>
3	Создание макета графического интерфейса	32	11.05.2025 – 14.05.2025	<i>Е</i>
4	Выбор технологий, применяемых при проектировании и разработке	18	15.05.2025 – 17.05.2025	<i>Е</i>
5	Реализация графического интерфейса	23	18.05.2025 – 20.05.2025	<i>Е</i>
6	Разработка основной части программы	40	21.05.2025 – 25.05.2025	<i>Е</i>
7	Оформление пояснительной записки	39	26.05.2025 – 30.05.2025	<i>Е</i>

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы *Е* 06.05.2025 Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент *Д.А.* 06.05.2025 Пивкин Д.А.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л.Р.* 16.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 123 листа, 42 рисунка, 9 источников, 4 приложения.

МИНИ-ПРИЛОЖЕНИЕ В TELEGRAM ДЛЯ МОНИТОРИНГА И АНАЛИЗА КРИПТОВАЛЮТНЫХ АКТИВОВ

Цель работы – разработать мини-приложение в Telegram, предоставляющее пользователям три ключевые функции: мониторинг курсов криптовалют с отображением актуальных цен, конвертер криптовалют, настройка уведомлений за выбранный промежуток времени.

Объектом разработки является процесс разработки финансовых Mini Apps в экосистеме Telegram, включая особенности WebApp API, методы интеграции криптовалютных API, способы реализации push-уведомлений.

Разработка проводилась в среде программирования Visual Studio Code. В качестве языков программирования были выбраны Python, JavaScript, SQL.

В результате проведенной работы разработано мини-приложение, удовлетворяющее поставленной задаче, а именно: интуитивно понятный пользовательский интерфейс и мониторинг актуальных цен на криптовалютные активы с персонализированными уведомлениями и конвертацией.

					ПГУ 1.090301.18.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Мини-приложение в Telegram для мониторинга и анализа криптовалютных активов Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Пивкин Д.А.		11.06.25					
Провер.		Гурин Е.И.		11.06.25			2	123	
Н.Контр		Бычков А.С.		11.06.25		ФВТ, каф ВТ гр. 21ВВП1			
Утверд.		Митрохин М.А.		11.06.25					

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Мини-приложение в Telegram для мониторинга и анализа криптовалютных активов	8
1.1. Анализ предметной области.....	8
1.1.1 Обзор аналогов	9
1.1.2 Анализ требований к разработке	17
1.1.3 Выбор архитектуры программы	18
1.1.4 Выбор языка программирования	20
1.1.5 Выбор среды разработки	22
1.2 Проектирование и разработка	35
1.2.1 Разработка backend-части	35
1.2.2 Разработка frontend-части	43
1.2.3 Создание мини-приложения в Telegram.....	49
1.3 Анализ результатов.....	51
2 Охрана труда	57
2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения.....	57
2.1.1 Оборудование рабочего места.....	58
2.1.2 Освещенность рабочего места	60
Заключение	61
Список использованных источников.....	62
Приложение А – Листинги backend-части	64
Приложение А.1 Файл «main.py»	64
Приложение А.2 Файл «router.py»	65
Приложение А.3 Файл «services.py»	66
Приложение А.4 Файл «models.py»	68
Приложение А.5 Файл «dto.py».....	71

Приложение А.6 Файл «tg.py».....	71
Приложение А.7 Файл «notifier.py»	72
Приложение А.8 Файл «requirements.txt».....	74
Приложение Б – Листинги frontend-части.....	76
Приложение Б.1 Файл «App.vue».....	76
Приложение Б.2 Файл «Navbar.vue»	77
Приложение Б.3 Файл «CryptoCard.vue».....	80
Приложение Б.4 Файл «HomeView.vue»	84
Приложение Б.5 Файл «ConverterView.vue»	88
Приложение Б.6 Файл «NotificationsView.vue»	100
Приложение Б.7 Файл «index.js».....	110
Приложение Б.8 Файл «main.js».....	111
Приложение Б.9 Файл «vite.config.js».....	112
Приложение В – UML-диаграммы.....	114
Приложение В.1 – UML-диаграмма вариантов использования	115
Приложение В.2 – UML-диаграмма развертывания	116
Приложение В.3 – UML-диаграмма классов	117
Приложение В.4 – UML-диаграмма компонентов	118
Приложение Г – Презентация.....	119

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

Бэкенд – это серверная часть веб-приложения или программного продукта, скрытая от пользователей. Она отвечает за обработку запросов, взаимодействие с базами данных, хранение и управление данными, а также логику работы приложения

Фронтенд – это клиентская, или видимая пользователю, часть веб-сайта или веб-приложения. Она включает в себя пользовательский интерфейс и все, с чем пользователь взаимодействует на странице

API (Application Programming Interface) – это набор правил и протоколов, которые определяют, как различные программные приложения могут взаимодействовать друг с другом

DeFi (Decentralized Finance) – это децентрализованная финансовая система, использующая блокчейн и смарт-контракты для предоставления финансовых услуг

Telegram Mini Apps – легковесные приложения, которые работают внутри мессенджера Telegram, не требуя отдельной установки на устройство

TON (The Open Network) – это децентрализованная блокчейн-платформа и сеть, разрабатываемая сообществом. Она представляет собой экосистему, включающую блокчейн, криптовалюту Toncoin, систему хранения данных TON Storage и другие компоненты

Visual Studio Code – это бесплатный, кроссплатформенный редактор исходного кода, разработанный Microsoft для Windows, macOS и Linux

Vue – прогрессивный JavaScript фреймворк для создания пользовательских интерфейсов. Он построен на стандартах HTML, CSS и JavaScript и предоставляет декларативную и компонентную модель программирования

Vite – это современный, быстрый и оптимизированный инструмент сборки веб-приложений, используемый для ускорения разработки и оптимизации статических ресурсов

					ПГУ 1.090301.18.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

С развитием цифровых технологий и распространением криптовалют потребность в удобных инструментах для мониторинга и анализа цифровых активов значительно возросла. Криптовалютный рынок характеризуется высокой волатильностью, что требует оперативного доступа к актуальной информации о курсах, капитализации, торговых объемах и других ключевых метриках. В этом контексте мини-приложения (Mini-app) внутри мессенджера Telegram представляют собой перспективное решение, объединяющее удобство использования, мгновенный доступ к данным и широкие функциональные возможности без необходимости установки дополнительных приложений.

Telegram, как одна из самых популярных платформ для общения, активно развивает экосистему ботов и мини-приложений, позволяя разработчикам создавать интерактивные сервисы прямо внутри мессенджера. Мини-приложение для мониторинга и анализа криптовалют может стать востребованным инструментом как для трейдеров, так и для обычных пользователей, желающих следить за динамикой своих инвестиций.

В данной работе рассматриваются предпосылки возникновения подобных решений, их аналоги, а также перспективы развития мини-приложений в Telegram как платформы для финансовых сервисов.

Криптовалютный рынок, зародившийся с появлением Bitcoin в 2009 году, прошел путь от нишевого технологического эксперимента до глобального финансового явления. Первые инструменты для мониторинга криптовалют были крайне примитивными: пользователи отслеживали курсы через форумы (например, Bitcointalk), а позже – через специализированные сайты вроде CoinMarketCap, появившегося в 2013 году.

					ПГУ 1.090301.18.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

С ростом числа криптовалют и увеличением торговых объемов возникла потребность в более удобных и оперативных способах получения информации.

Мини-приложения в Telegram предлагают компромисс: они работают внутри мессенджера, не требуют установки, обладают почти таким же функционалом, как полноценные приложения, и могут интегрироваться с другими сервисами через API.

Современный крипторынок представляет собой высокодинамичную среду, где цены активов могут изменяться на 10-20% в течение суток. Для эффективного участия в этом рынке трейдеры, инвесторы и даже новички нуждаются в оперативном доступе к актуальной информации о курсах криптовалют, удобных инструментах конвертации и своевременных уведомлениях о значимых изменениях.

Telegram Mini Apps открывают новые возможности для создания финансовых сервисов, сочетающих удобство мессенджера с функциональностью полноценных приложений.

					<i>ПГУ 1.090301.18.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Мини-приложение в Telegram для мониторинга и анализа криптовалютных активов

1.1 Анализ предметной области

Разработка Mini App для крипторынка находится на стыке нескольких областей: финтех, веб-разработка, мобильные технологии и безопасность. В финтехе это работа с финансовыми данными и алгоритмами конвертации, в веб-разработке – создание интерфейсов и интеграция с API, в мобильных технологиях – адаптация под сенсорные устройства, а в безопасности – защита пользовательских данных. Ключевые вызовы включают обеспечение актуальности данных при ограничениях API, создание удобного интерфейса для базовых финансовых операций и реализацию мгновенных уведомлений без задержек.

Актуальность темы обусловлена несколькими факторами. Во-первых, крипторынок демонстрирует значительный рост: его капитализация превышает \$3,4 трлн, а ежедневный оборот составляет десятки миллиардов долларов. Во-вторых, пользователи, особенно трейдеры, нуждаются в мобильных решениях, при этом многие используют несколько приложений одновременно для мониторинга рынка, а push-уведомления о ключевых изменениях курса значительно повышают удобство. В-третьих, Telegram Mini Apps предлагают готовую аудиторию из более чем 1 млрд пользователей, встроенные платежные решения и потенциал вирусного распространения через чаты.

В итоге мини-приложение для мониторинга и анализа криптовалютных активов предоставляет пользователям мгновенный доступ к актуальным курсам без необходимости использования сложных платформ, упрощает конвертацию между криптовалютами и предлагает интуитивно понятный интерфейс, оптимизированный для мобильных устройств через Telegram.

					ПГУ 1.090301.18.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1.1.1 Обзор аналогов

Помимо данного приложения существуют различные аналоги как в самом мессенджере Telegram, так и за его пределами. Ниже будут представлены прямые аналоги разрабатываемого мини-приложения.

Первым из подобных аналогов выступает @Wallet – это мини-приложение и бот в Telegram, интегрированный с блокчейном TON (The Open Network). Позволяет пользователям хранить, переводить и отслеживать криптовалютные активы. На рисунке 1 представлена главная страница мини-приложения «Кошелёк», а на рисунке 2 продемонстрированы популярные криптовалюты, динамика их цен и процент роста соответственно.

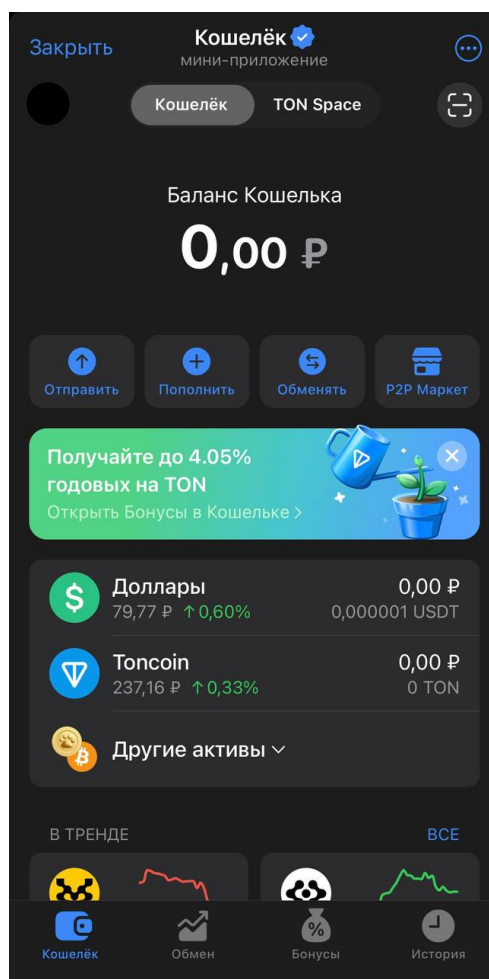


Рисунок 1 – Главная страница мини-приложения @Wallet

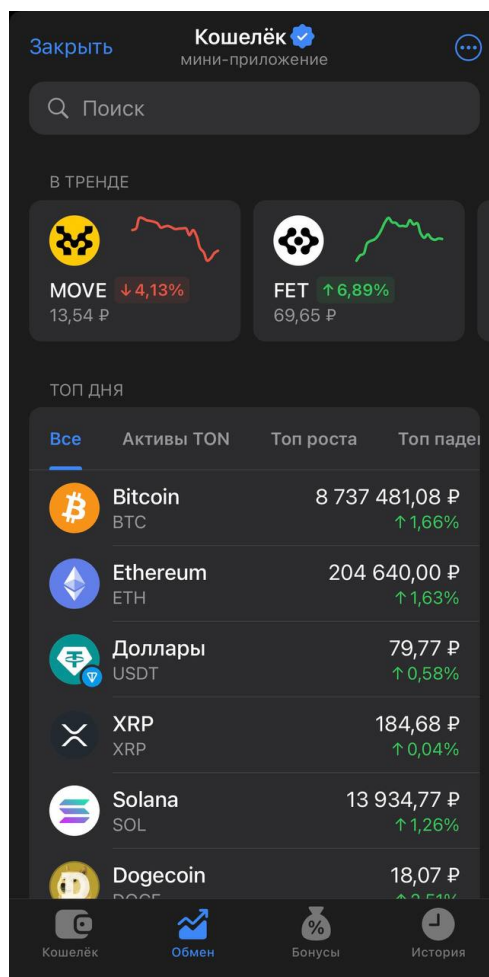


Рисунок 2 – Обзор популярных криптовалют в @Wallet

Достоинства данного приложения заключаются в глубокой интеграции с Telegram (работает без установки внешних приложений, использует встроенные платежи Telegram), поддержке мультивалютности (позволяет управлять разными криптоактивами в одном интерфейсе), быстрых транзакциях (благодаря блокчейну TON комиссии низкие, а переводы почти мгновенные), доступу к DeFi – поддерживает стейкинг, обмен токенов и подключение к децентрализованным приложениям.

Недостатки: ограниченная аналитика (фокусируется на транзакциях, а не на глубоком анализе рынка), зависимость от TON (не все криптотрейдеры используют этот блокчейн), нет сложных графиков – инструменты для технического анализа минимальны.

Вывод: хорош для быстрых операций, но слаб для профессионального трейдинга.

Вторым аналогом выступает мини-приложение @CryptoBot – один из самых популярных криптоботов в Telegram, позволяющий торговать, отслеживать курсы и получать уведомления о рыночных изменениях. Ниже на рисунке 3 показана главная страница данного бота. Также на рисунке 4 представлена биржа для торговли криптовалютой внутри приложения.

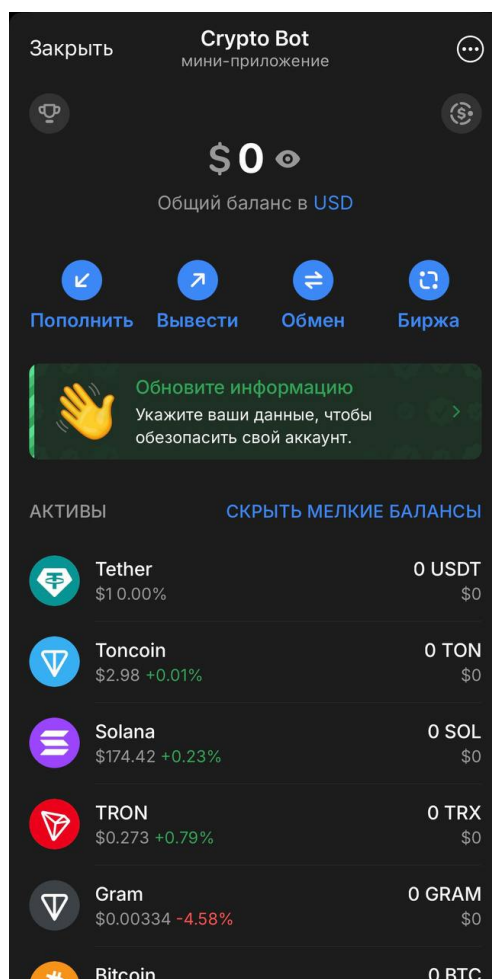


Рисунок 3 – Главная страница мини-приложения @CryptoBot

Данное мини-приложение представляет широкий функционал (трейдинг, портфельный трекинг, автоматические уведомления), поддержку P2P-обменов (можно покупать/продавать криптовалюту без выхода из Telegram), интеграцию с биржами (поддерживает Binance, KuCoin и другие), гибкие API-настройки (можно подключать собственные торговые стратегии).

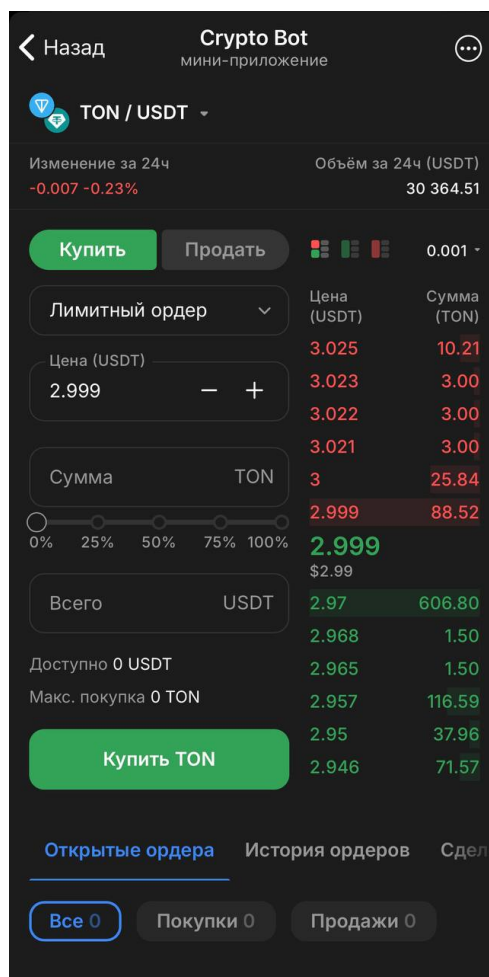


Рисунок 4 – Биржа для торговли криптовалютой в @CryptoBot

К недостаткам относятся: сложный интерфейс (новичкам может быть трудно разобраться), нет подробного технического анализа (только базовые графики), комиссии выше среднего (особенно при P2P-сделках).

Таким образом, данное мини-приложение довольно универсальное, но требует доработки аналитического функционала.

Следующий аналог для разрабатываемого проекта – @CoinGeckoBot. Данный бот является официальным продуктом CoinGecko, предоставляющим актуальные данные о курсах, капитализации и трендах крипторынка. В отличие от предыдущих аналогов этот бот не имеет своего мини-приложения и взаимодействие с ним происходит через команды внутри чата. На рисунке 5 показано приветственное сообщение при взаимодействии с ботом с помощью команды /start. Также на рисунке 6 произведен переход на веб-версию CoinGecko внутри Telegram.

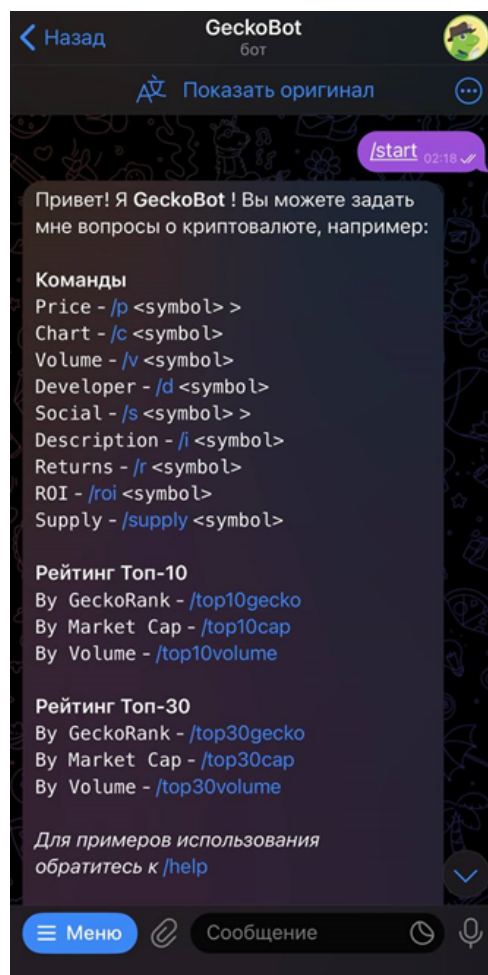


Рисунок 5 – Приветственное сообщение с демонстрацией функционала от бота GeckoBot

Достоинством данного продукта является обновление данных в реальном времени (агрегирует информацию с множества бирж), глубокая аналитика (включает исторические данные, графики, объемы торгов), удобные команды (можно быстро получать данные по конкретным монетам), бесплатный доступ (в отличие от некоторых платных аналогов).

Все вышеперечисленные преимущества нивелируются рядом недостатков, таких как: отсутствие торгового функционала (только мониторинг), ограниченные настройки уведомлений (нельзя гибко настраивать оповещения о изменении цен на желаемые монеты), нет интеграции с кошельками (нельзя управлять активами).

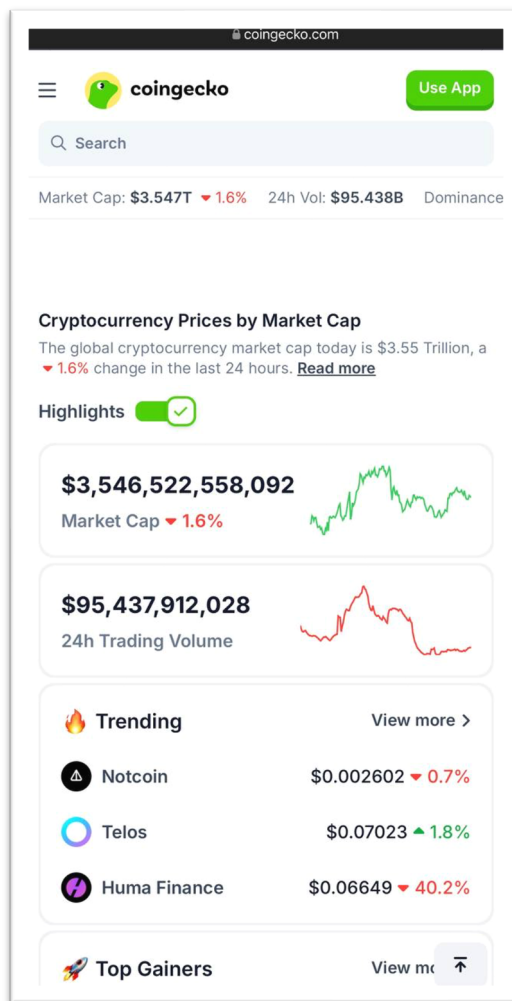


Рисунок 6 – Переход на веб-версию CoinGecko прямо в Telegram

При рассмотрении всех вышеописанных преимуществ и недостатков был сделан вывод, что данный бот весьма полезен для мониторинга и анализа всех популярных криптовалют, но в нем отсутствует возможность конвертации и торговли.

Также кроме прямых аналогов существуют и косвенные аналоги, которые в свою очередь представляются в виде веб-сайтов.

Первым из таких косвенных аналогов является CoinMarketCap – один из крупнейших агрегаторов данных о криптовалютах с веб-версией и мобильным приложением. Главная страница данного веб-сайта продемонстрирована на рисунке 7.

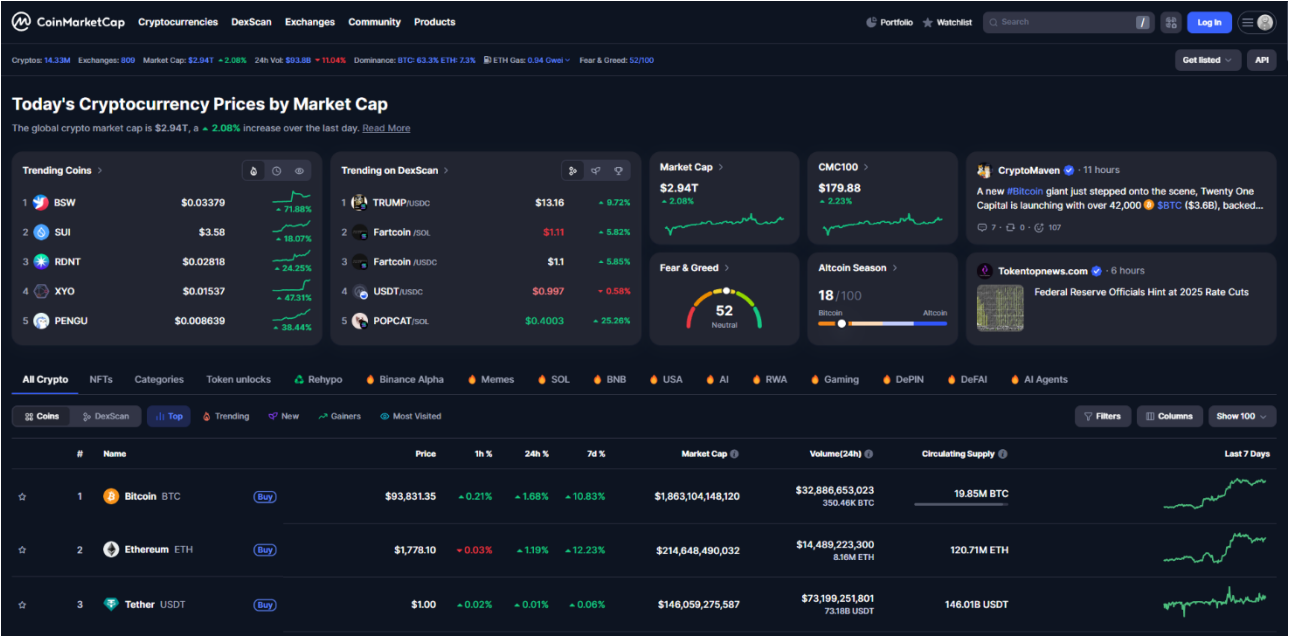


Рисунок 7 – Главная страница веб-сайта CoinMarketCap

К достоинствам данного веб-ресурса можно отнести ряд полезного функционала и привилегий, к которым относятся: самая полная база данных (охватывает тысячи активов), расширенная аналитика (графики, индексы, метрики DeFi), портфельный трекинг (можно следить за своими инвестициями).

Также как и у всех вышеописанных аналогов, у данного веб-сайта присутствуют и недостатки, к которым можно отнести: отсутствие торговли внутри платформы (только информация о динамике цен на криптовалюту, графики и т.д.), реклама и платные функции (некоторые данные доступны только в Pro-версии).

Таким образом, можно отметить, что данный агрегатор является лучшим на рынке для анализа текущего курса и динамики цен на криптовалюту, но отсутствие интеграции в простой мессенджер, на подобии Telegram делает данный веб-сайт не удобным для использования на мобильных устройствах.

Ещё одним из косвенных аналогов для разрабатываемого мини-приложения выступает Binance – крупнейшая криптобиржа с собственным приложением, включающим торговлю, аналитику и кошелёк. На рисунке 8

показана главная страница данного ресурса.

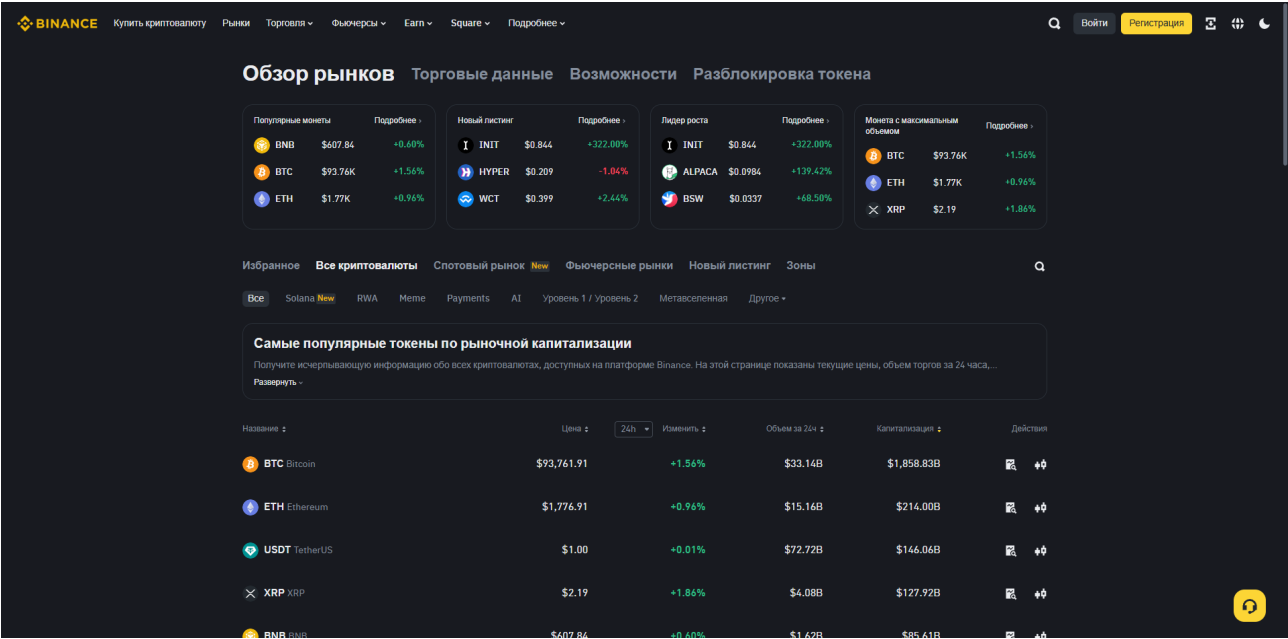


Рисунок 8 – Главная страница веб-сайта Binance

Преимуществом данного веб-сайта является следующий функционал: полноценный трейдинг (спот, фьючерсы, стейкинг), глубокая аналитика (TradingView-графики, индикаторы), высокая ликвидность (лучшие цены на рынке).

Недостатками Binance выступают следующие проблемы: сложность для новичков (требуется обучения), регуляторные ограничения (в некоторых странах доступ ограничен, в том числе регистрация в РФ запрещена на данный момент).

Вывод: Binance является мощной платформой для аналитики и торговли, но не заточен под Telegram, из-за чего обучение всем возможностям данного сервиса может быть очень затруднительным для новичков.

Подводя итог, можно отметить, что существующие решения не предлагают комплексного подхода, сочетающего удобный интерфейс, полный набор инструментов и глубокую интеграцию с Telegram.

1.1.2 Анализ требований к разработке

Разработка мини-приложения для мониторинга и анализа криптовалютных активов в Telegram требует тщательного анализа функциональных и нефункциональных требований, а также учета особенностей целевой аудитории и технологического стека. Ниже представлен детальный разбор ключевых аспектов, которые были учтены на этапе проектирования и реализации.

Функциональные требования определяют основные возможности, которые должно предоставлять приложение пользователям. Для данного мини-приложения они включают:

- отображение актуальных курсов криптовалют. Приложение должно в реальном времени показывать текущие курсы популярных криптовалют, таких как Bitcoin (BTC), Ethereum (ETH), Tether (USDT), Solana (SOL) и Tron (TRX). Данные должны обновляться автоматически каждые 60 секунд с указанием процента изменения цены за сутки;

- конвертер валют. Пользователи должны иметь возможность конвертировать одну криптовалюту в другую. Интерфейс конвертера должен быть интуитивно понятным, с возможностью ввода суммы и выбора исходной и целевой валют;

- уведомления. Приложение должно поддерживать настройку уведомлений о изменениях курсов. Пользователи могут выбирать отслеживаемые криптовалюты и частоту уведомлений (каждую минуту, каждый час, раз в день). Уведомления должны быть своевременными и содержать актуальную информацию;

- интерфейс управления. Пользовательский интерфейс должен быть разделен на логические разделы: "Курсы валют", "Конвертер" и "Уведомления". Это обеспечит удобную навигацию и быстрый доступ к нужным функциям.

Нефункциональные требования описывают, как система должна работать, и включают следующие аспекты:

- производительность. Приложение должно быстро загружать и отображать актуальные данные. Задержки при обновлении курсов или конвертации валют должны быть минимальными;

- надежность. Система должна быть устойчива к сбоям и обеспечивать бесперебойную работу. В случае ошибок при получении данных (например, из-за проблем с API) приложение должно корректно обрабатывать такие ситуации и уведомлять пользователя;

- безопасность. Все пользовательские данные, включая настройки уведомлений, должны храниться безопасно. Приложение не должно запрашивать или хранить конфиденциальную информацию без необходимости;

- масштабируемость. Архитектура приложения должна позволять легко добавлять новые криптовалюты или функции в будущем без значительных изменений в коде;

- удобство использования. Интерфейс должен быть простым и понятным даже для пользователей с минимальным опытом работы с криптовалютами. Важно обеспечить адаптивность дизайна для разных устройств и динамическую смену темы в зависимости от выбранной темы внутри Telegram.

1.1.3 Выбор архитектуры программы

Клиентская часть мини-приложения (Frontend) отвечает за отображение курсов криптовалют в виде списка, предоставляет функционал конвертации одной валюты в другую и обратно, а также позволяет пользователям настраивать уведомления об изменениях цен. Для реализации этих задач используются современные технологии, такие как Vue.js в связке с JavaScript, что обусловлено простотой и удобством этого фреймворка для веб-разработки, а также наличием большого количества готовых решений и библиотек.

Интеграция с Telegram обеспечивается за счет официальной библиотеки Telegram WebApp, которая упрощает встраивание приложения в мессенджер и предоставляет доступ к данным пользователя без необходимости ввода логинов и паролей [1]. Для работы с API был выбран Axios вместо нативного fetch из-за его удобства. Axios предоставляет более понятный синтаксис, автоматическую обработку ошибок, поддержку интерсепторов (например, для добавления заголовков авторизации) и обратную совместимость с устаревшими браузерами.

Серверная часть (Backend) выполняет ключевые функции, включая получение актуальных данных о курсах криптовалют, хранение пользовательских настроек и отправку уведомлений. В качестве основного языка разработки выбран Python благодаря его простоте, скорости развертывания и удобству для быстрой разработки. Для хранения данных используется SQLite – легковесная и гибкая база данных, не требующая сложной настройки, что делает ее оптимальным решением для данного проекта.

Взаимодействие с внешними сервисами играет важную роль в работе приложения. Для получения данных о криптовалютах задействуется CoinGecko API – бесплатный и надежный источник информации, поддерживающий более 10 000 активов, но имеющий ограничение в 50 запросов в минуту [2]. Отправка уведомлений пользователям реализована через Telegram Bot API, который обеспечивает простую и эффективную интеграцию с мессенджером.

В перспективе проект может быть улучшен за счет подключения дополнительных источников данных, таких как Binance и CoinMarketCap, что повысит точность и актуальность информации. Также рассматривается возможность создания PWA-версии приложения для работы вне Telegram, а также добавления аналитических функций, включая прогнозирование цен и расширенную визуализацию данных с помощью графиков.

Диаграмма вариантов использования мини-приложения должна выделять три основных сценария: мониторинг курсов, конвертацию валют и

управление уведомлениями – что полностью соответствует первоначальной концепции мини-приложения. Бот выступает отдельным актором, ответственным за push-уведомления, что подчеркивает автономность системы оповещений от основного интерфейса. Разделение на выбор исходной/целевой валюты демонстрирует поддержку крипто-крипто пар. На выходе получается UML-диаграмма, показанная на рисунке 9.

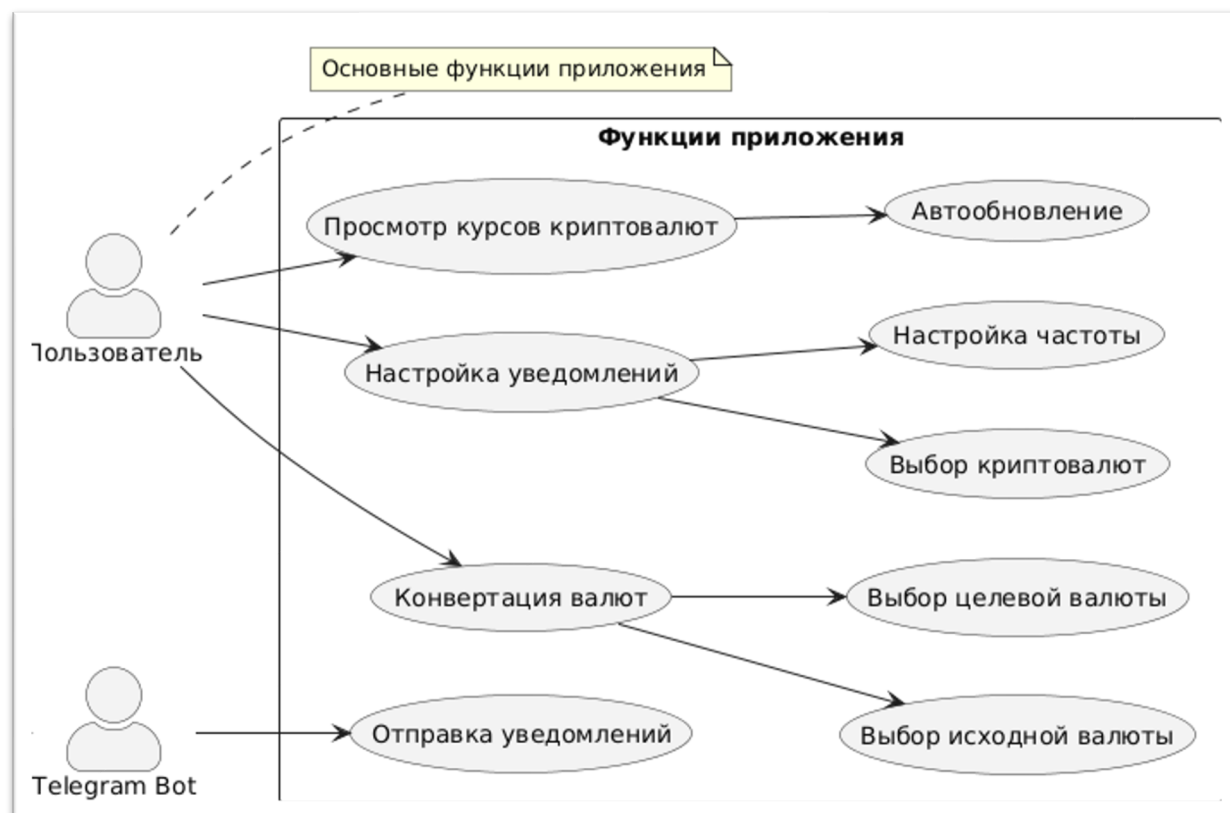


Рисунок 9 – UML-диаграмма вариантов использования мини-приложения

1.1.4 Выбор языка программирования

При разработке мини-приложения для мониторинга и анализа криптовалютных активов выбор языков программирования был обусловлен спецификой задач, требованиями к производительности, простотой разработки и интеграции с экосистемой Telegram [2]. Было рассмотрено обоснование выбора языков для клиентской и серверной частей приложения. Для реализации клиентской части была выбрана связка Vue.js и JavaScript. Этот выбор обусловлен несколькими ключевыми факторами.

Во-первых, Vue.js является прогрессивным фреймворком, который сочетает в себе простоту освоения и высокую производительность, что критично для мини-приложения, работающего внутри Telegram. Vue.js предлагает реактивность данных и компонентный подход, что упрощает разработку динамических интерфейсов, таких как списки курсов криптовалют и формы конвертации. Во-вторых, экосистема Vue.js включает множество готовых решений и библиотек (например, Vuex для управления состоянием), что ускоряет разработку. JavaScript, как базовый язык, обеспечивает совместимость с Web API Telegram и браузерной средой, в которой работает мини-приложение [3]. Альтернативы, такие как React или Angular, также могли быть использованы, но Vue.js был выбран из-за более низкого порога входа и лучшей интеграции с Telegram WebApp.

Для работы с API был выбран нативный fetch вместо Axios, поскольку он является стандартным API современных браузеров и не требует подключения дополнительных библиотек [3]. Fetch предоставляет простой и понятный интерфейс для выполнения HTTP-запросов, а также поддерживает промисы, что позволяет удобно работать с асинхронными операциями. Несмотря на то, что fetch требует ручной обработки некоторых аспектов (например, проверки статуса ответа и преобразования данных), его использование снижает зависимость проекта от сторонних библиотек. Это особенно важно для мини-приложения в Telegram, где критична скорость загрузки и работы интерфейса. Кроме того, современные браузеры, включая мобильные, имеют полноценную поддержку fetch API, что обеспечивает стабильную работу приложения у большинства пользователей [4].

Серверная часть приложения была реализована на Python. Этот выбор обусловлен следующими преимуществами языка. Во-первых, Python обладает простым и читаемым синтаксисом, что ускоряет разработку и уменьшает количество ошибок.

Во-вторых, Python имеет богатую экосистему библиотек для работы с API (например, requests), базами данных и асинхронными задачами, что критично для получения данных о курсах криптовалют и отправки уведомлений. В-третьих, Python легко интегрируется с Telegram Bot API, что упрощает реализацию push-уведомлений [5]. Альтернативы, такие как Node.js или Go, также могли быть рассмотрены, но Python был выбран из-за его популярности в сфере финтеха и наличия готовых решений для работы с криптовалютными API (например, библиотеки для CoinGecko).

Для хранения данных была выбрана SQLite – легковесная встраиваемая СУБД, которая не требует отдельного сервера и легко настраивается. Это оптимальное решение для небольшого проекта, где не требуется масштабируемость и распределенная архитектура. SQLite также хорошо интегрируется с Python через стандартную библиотеку sqlite3 или ORM-решения типа SQLAlchemy. Альтернативы, такие как PostgreSQL или MongoDB, были бы избыточны для данного проекта на начальном этапе.

Для работы с криптовалютными данными был выбран CoinGecko API, так как он предоставляет бесплатный доступ к информации о тысячах активов и имеет достаточно либеральные ограничения на запросы (50 запросов в минуту) [6]. Telegram Bot API был использован для отправки уведомлений, так как он является стандартным инструментом для интеграции с мессенджером и поддерживает все необходимые функции.

1.1.5. Выбор среды разработки

При разработке мини-приложения для Telegram, предназначенного для мониторинга криптовалют, ключевым инструментом стал Visual Studio Code (VS Code) – мощный, гибкий и высокопроизводительный редактор кода, который идеально подходит для подобных проектов. Его выбор обусловлен целым рядом преимуществ, делающих его предпочтительным не только для этого бота, но и для многих других задач в области программирования.

Visual Studio Code выделяется среди аналогов благодаря своей легкости и высокой скорости работы, что особенно важно при работе над небольшими, но требовательными к отзывчивости интерфейса проектами, такими как Telegram-бот. В отличие от более тяжеловесных сред разработки, таких как PyCharm или IntelliJ IDEA, VS Code запускается практически мгновенно и не нагружает систему даже при длительной работе, что позволяет разработчику сосредоточиться на коде, а не на ожидании загрузки инструментов.

Кроссплатформенность VS Code – еще одно важное преимущество, поскольку он стабильно работает на Windows, macOS и Linux, обеспечивая идентичный опыт разработки вне зависимости от операционной системы. Это особенно ценно, если над проектом работает несколько человек с разными устройствами или если разработчик тестирует бота в разных средах.

Гибкость и расширяемость VS Code делают его универсальным решением для самых разных задач. Благодаря встроенному магазину расширений (Marketplace) редактор можно адаптировать под любые нужды – будь то Python, JavaScript, Go или другие языки программирования. Для данного проекта особую ценность представляют расширения, связанные с Python, такие как Pylance и Python Extension Pack, которые обеспечивают интеллектуальное автодополнение кода, подсветку синтаксиса, проверку ошибок (linting) и удобную работу с виртуальными окружениями.

Интегрированный терминал в VS Code значительно ускоряет процесс разработки, позволяя запускать скрипты, серверы и команды Git прямо из редактора, без необходимости переключаться между окнами. Это особенно удобно при работе с Telegram-ботом, где часто требуется тестировать изменения в реальном времени или проверять ответы API. Поддержка различных оболочек, включая PowerShell, Bash и WSL, делает терминал универсальным инструментом для любых задач.

Отдельного внимания заслуживает встроенная поддержка Git, которая упрощает контроль версий и совместную работу над проектом. VS Code визуализирует изменения в коде, позволяет быстро фиксировать правки (commit) и отправлять их в репозиторий (push), а также решать конфликты слияния прямо в интерфейсе редактора. Это избавляет от необходимости использовать дополнительные программы вроде GitKraken или Sourcetree.

Интеллектуальное автодополнение кода (IntelliSense) – одна из ключевых функций, ускоряющих написание бота. Оно не только подсказывает синтаксис, но и анализирует контекст, предлагая релевантные варианты методов, переменных и параметров функций. Это особенно полезно при работе с библиотеками python-telegram-bot и aiogram, где важно точно соблюдать структуру запросов к Telegram API.

Встроенный отладчик – еще один аргумент в пользу VS Code, поскольку он позволяет находить и исправлять ошибки прямо в редакторе. Для Telegram-бота это критически важно: можно пошагово выполнять код, проверять значения переменных и анализировать логику работы без необходимости запускать внешние инструменты отладки.

Почему VS Code идеально подходит именно для разработки Telegram-бота? Во-первых, он обеспечивает превосходную поддержку Python – основного языка, на котором пишутся подобные боты. Расширения для Python превращают VS Code в почти полноценную IDE, предлагая все необходимые инструменты для комфортной работы. Во-вторых, Telegram Bot API активно использует JSON для обмена данными, и VS Code предоставляет удобную подсветку и валидацию JSON, что упрощает чтение и редактирование ответов API.

Если в будущем потребуется добавить аналитические функции, такие как построение графиков или прогнозирование курсов, VS Code поддерживает Jupyter Notebook прямо в интерфейсе. Это открывает возможности для сложного анализа данных без переключения между приложениями.

Наконец, работа с конфигурационными файлами (config.py, .env, YAML) в VS Code максимально удобна благодаря подсветке синтаксиса и автоматической проверке ошибок. Это снижает риск опечаток и ускоряет настройку бота под разные сценарии использования.

Таким образом, Visual Studio Code – это не просто текстовый редактор, а мощная и универсальная среда разработки, которая сочетает в себе легкость, производительность и богатый функционал. Его выбор для создания Telegram-бота оказался полностью оправданным, так как он позволил сосредоточиться на логике приложения, минимизировав время на настройку инструментов и решение технических проблем.

Данный текстовый редактор, почти что близкий к полноценной IDE, бесплатен и скачать его можно с официального сайта, который показан на рисунке 10.

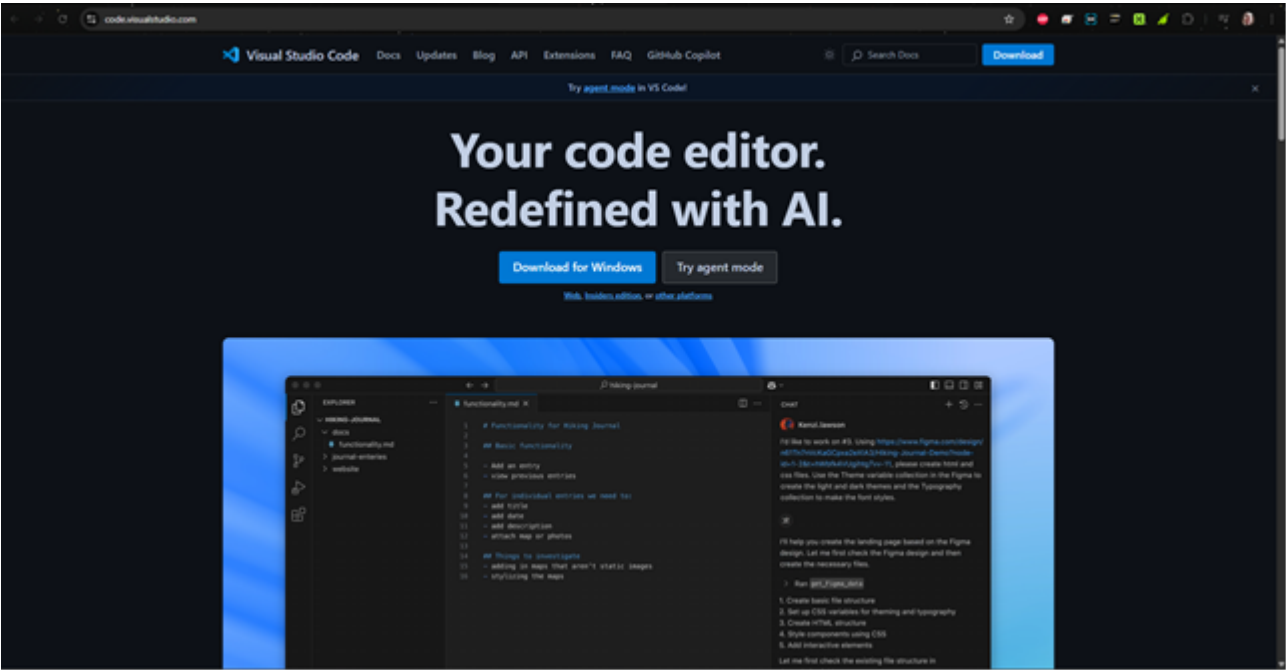


Рисунок 10 – Официальный сайт VS Code

Страница среды VS Code на сайте, помимо ссылки для загрузки исполняемого файла установки, содержит описание её функционала, информацию о системных требованиях, ссылки на техническую документацию, ответы на частые вопросы и т.п.

Чтобы начать процесс загрузки данного текстового редактора требуется нажать «Download for Windows», что показано также на рисунке 10.

После нажатия кнопки «Download for Windows» начинается загрузка исполняемого файла для установки VS Code, как показано на рисунке 11.

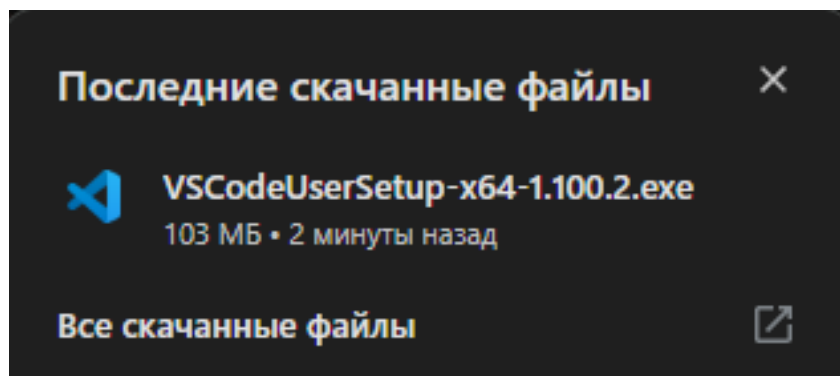


Рисунок 11 – Успешное скачивание исполняемого файла с сайта

После запуска установочного файла открывается окно Мастера установки VS Code. Для продолжения установки требуется нажать «Далее». Окно Мастера установки показано на рисунке 12. В процессе также потребуется ознакомиться с условиями лицензионного соглашения и принять их.

На следующем этапе можно создать значок на Рабочем столе, но тут необходимо выбрать пункт «Добавить в PATH», что позволит нашей операционной системе находить и запускать этот файл из любой директории в командной строке или терминале, без необходимости указывать полный путь к файлу. Данное окно показано на рисунке 13.

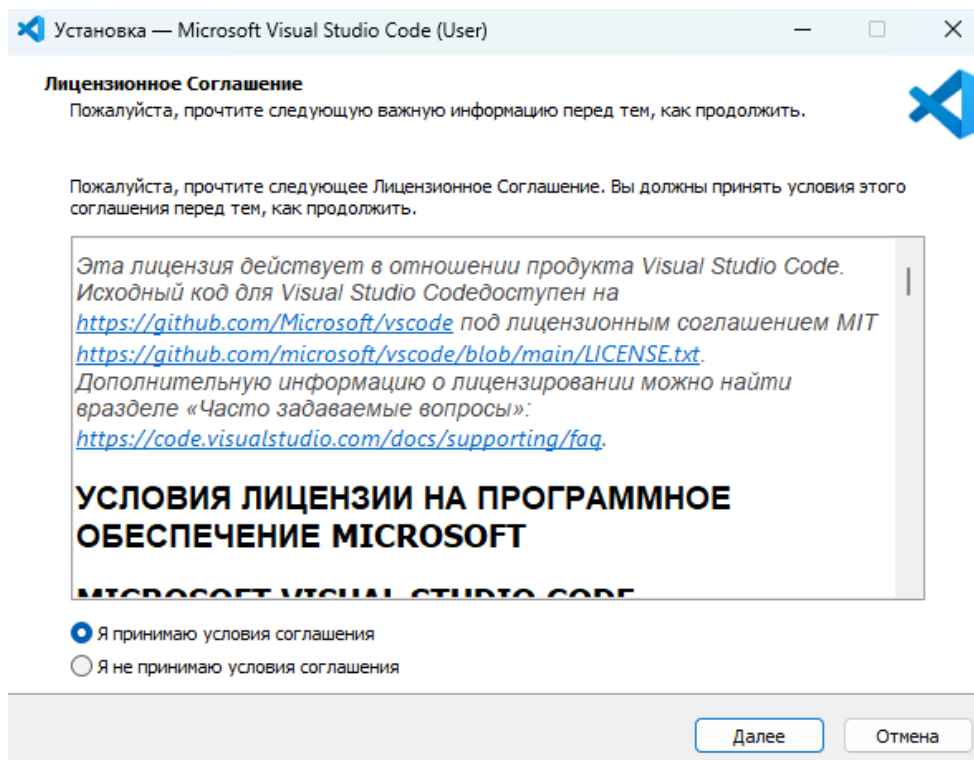


Рисунок 12 – Принятие условий лицензионного соглашения

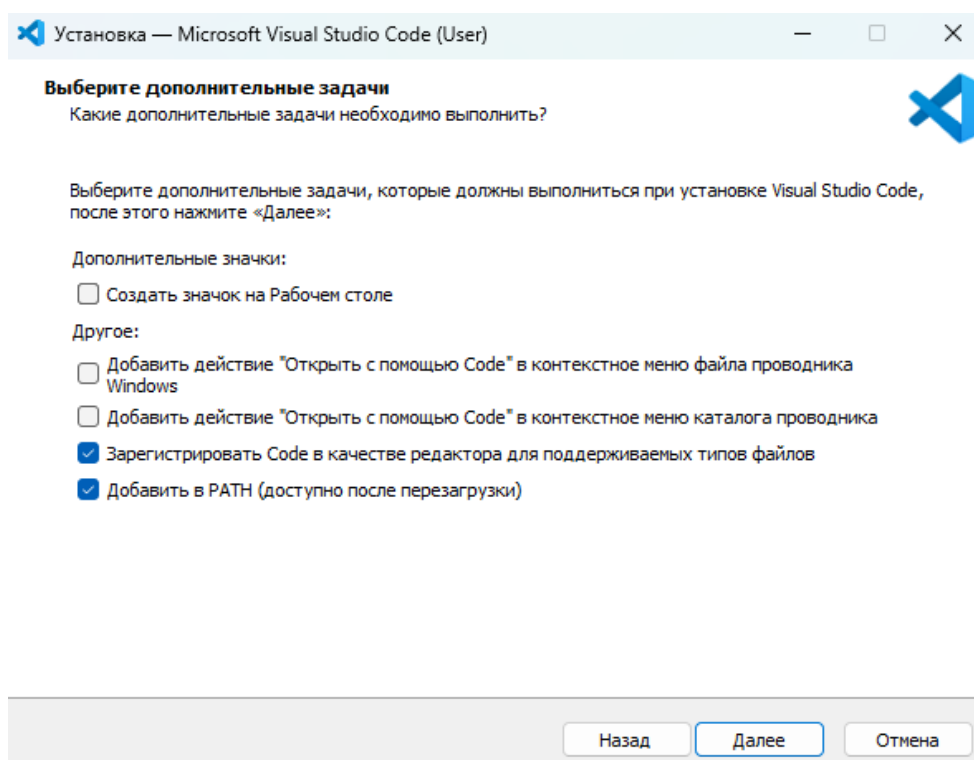


Рисунок 13 – Выбор дополнительных задач при установке

В окне, показанном на рисунке 14, предлагается проверить все дополнительные опции и начать процесс установки.

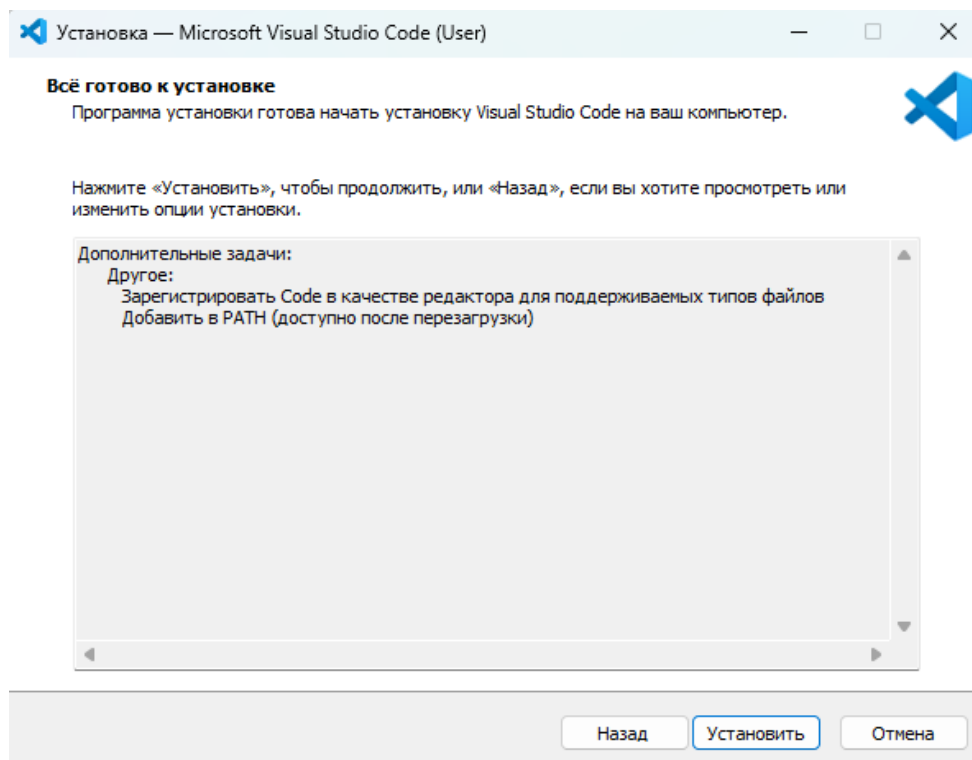


Рисунок 14 – Просмотр опций установки

После успешной установки будет предложено запустить Visual Studio Code, как показано на рисунке 15.

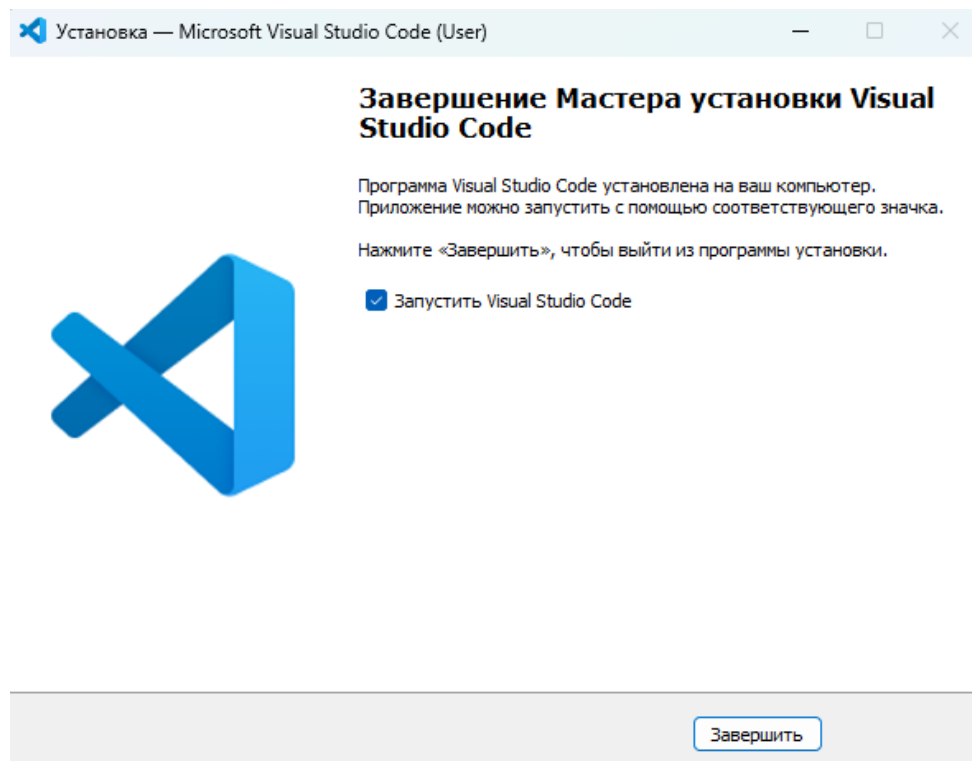


Рисунок 15 – Завершение установки VS Code

На рисунке 16 показано Главное окно, отрывающееся после запуска выбранного текстового редактора.

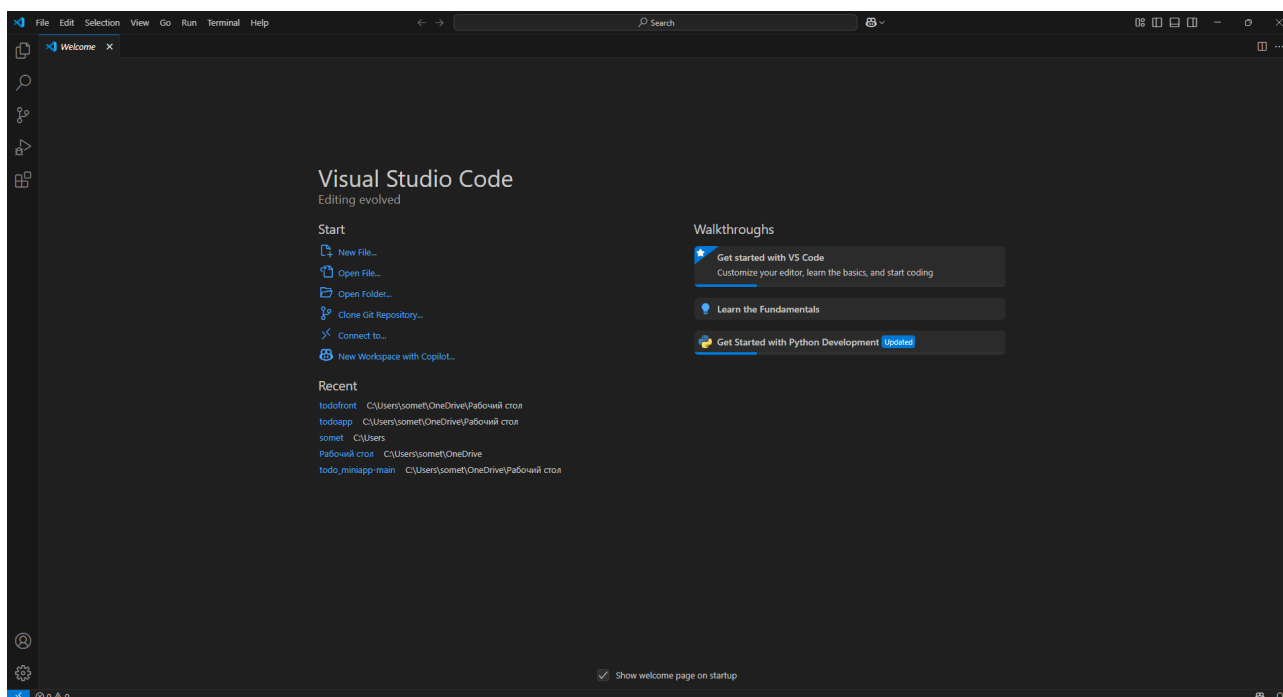


Рисунок 16 – Окно Visual Studio Code

Для настройки среды программирования VS Code требуется создать новый проект через кнопку на главном окне «New File...», либо же перейти во вкладку «File» и уже там выбрать тот же пункт, как показано на рисунке 17.

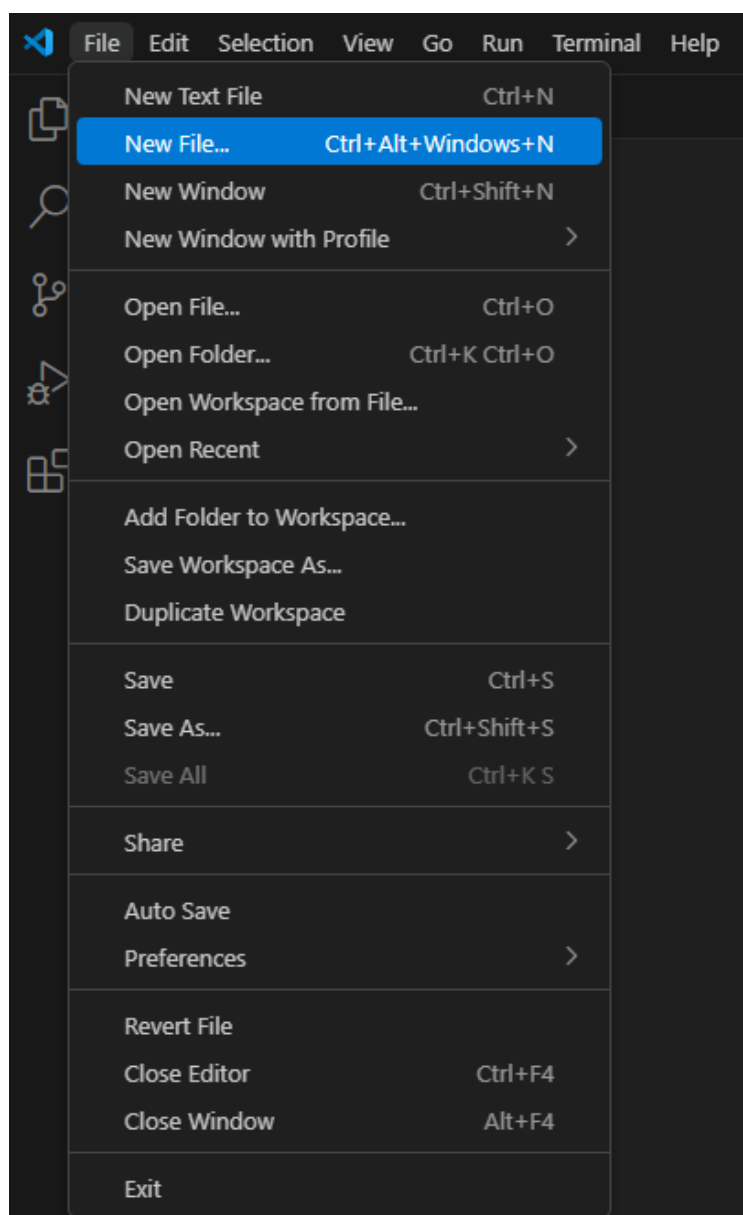


Рисунок 17 – Создание нового проекта через вкладку «File»

Visual Studio Code предлагает удобную боковую панель с несколькими вкладками, каждая из которых выполняет свою функцию в процессе разработки. Рассмотрим подробно на рисунке 18 назначение и возможности каждой из них для разработки мини-приложения в Telegram и в целом.

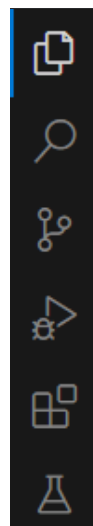


Рисунок 18 – Боковая панель VS Code

Вкладка Explorer – это центральный узел для навигации по файлам и папкам проекта. Она отображает структуру каталогов в виде дерева, что особенно полезно при работе с большими проектами. Также она позволяет открывать, переименовывать, удалять и создавать новые файлы и папки прямо из интерфейса, поддерживает одновременное открытие нескольких папок (Workspaces), что удобно, если проект состоит из нескольких модулей или интеграций, открывает дополнительные опции, такие как копирование пути, поиск в файле или запуск в терминале. Данная вкладка показана на рисунке 19.

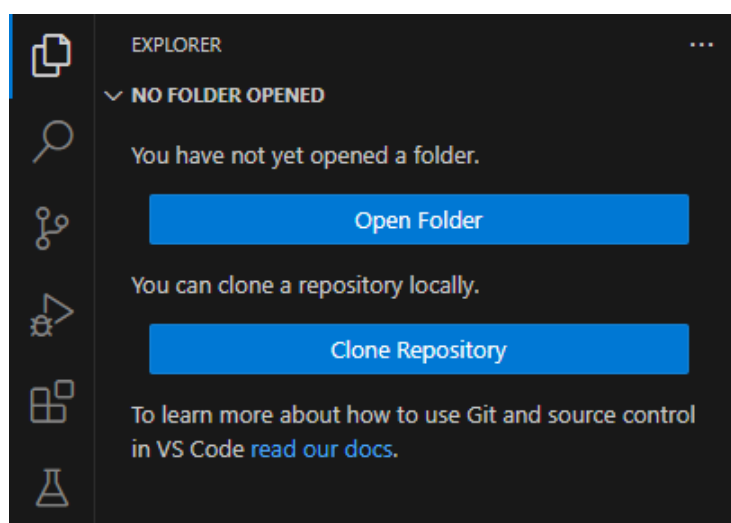


Рисунок 19 – Вкладка Explorer

Вкладка Search, которая показана на рисунке 20, предоставляет мощный инструмент для поиска текста в файлах проекта, включая поддержку регулярных выражений и фильтрацию по типам файлов. Она позволяет искать конкретные функции, переменные или сообщения бота во всех файлах проекта. Например, быстро найти все места, где используется команда /start, массово изменять текст (например, если нужно переименовать переменную во всех файлах). Также поиск можно ограничить определенными файлами (например, только *.py или *.json).

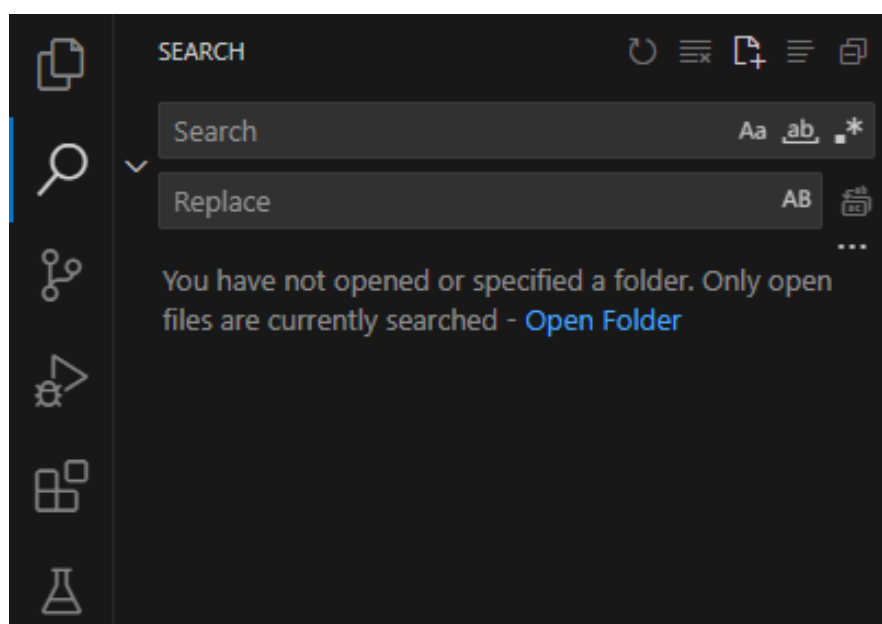


Рисунок 20 – Вкладка Search

Вкладка Source Control, которая продемонстрирована на рисунке 21, интегрирована с Git и другими системами контроля версий, предоставляя удобный интерфейс для управления изменениями. Она показывает измененные, добавленные и удаленные файлы, Позволяет фиксировать изменения (commit), писать сообщения и отправлять их в удаленный репозиторий (push), поддерживает создание и переключение между ветками прямо из VS Code.

Для создания мини-приложения в Telegram это критически важно, так как данная вкладка позволяет откатывать ошибочные изменения, упрощает совместную работу над кодом, дает возможность вести историю изменений.

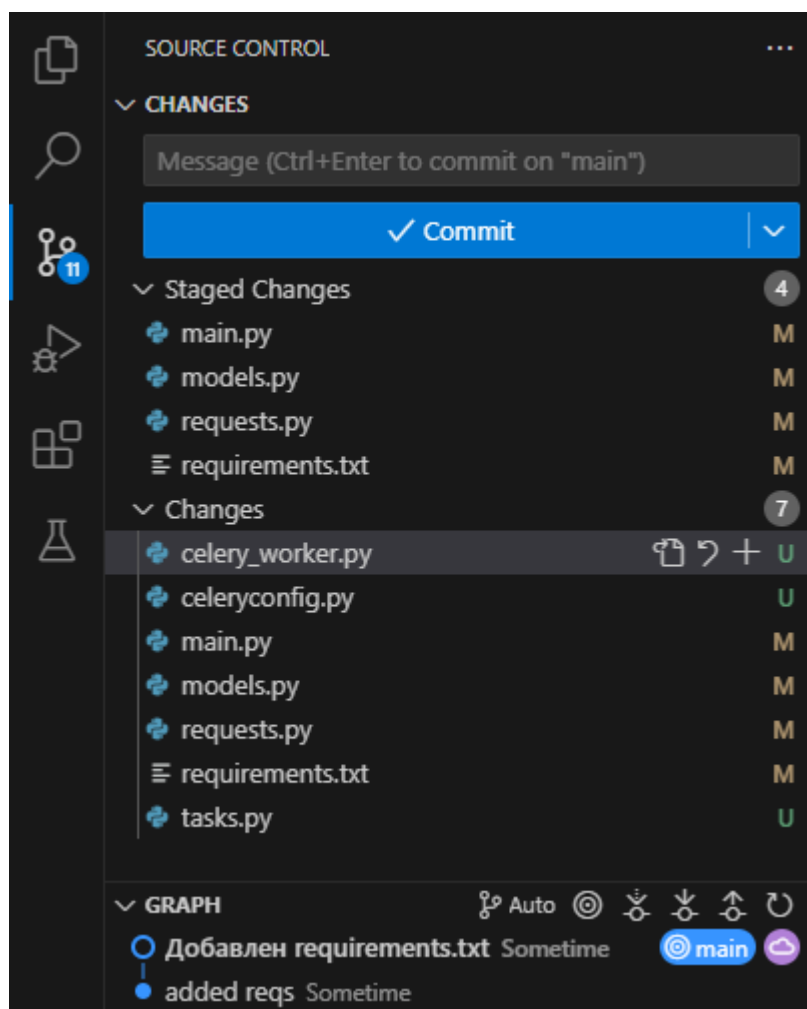


Рисунок 21 – Пример работы вкладки Source Control при отправке проекта в репозиторий GitHub

Вкладка Run and Debug – это инструмент для тестирования и отладки кода. Она позволяет выполнять код без ручного ввода команд в терминале, поддерживает точки останова (breakpoints), пошаговое выполнение (Step Over, Step Into) и просмотр переменных, а также помогает создавать профили для разных сценариев запуска (например, с разными аргументами). Данная вкладка показана на рисунке 22.

В контексте выполнения данного проекта эта вкладка особенно помогает отлавливать ошибки в обработчиках сообщений, анализировать работу API-интеграций, тестировать новые функции перед деплоем.

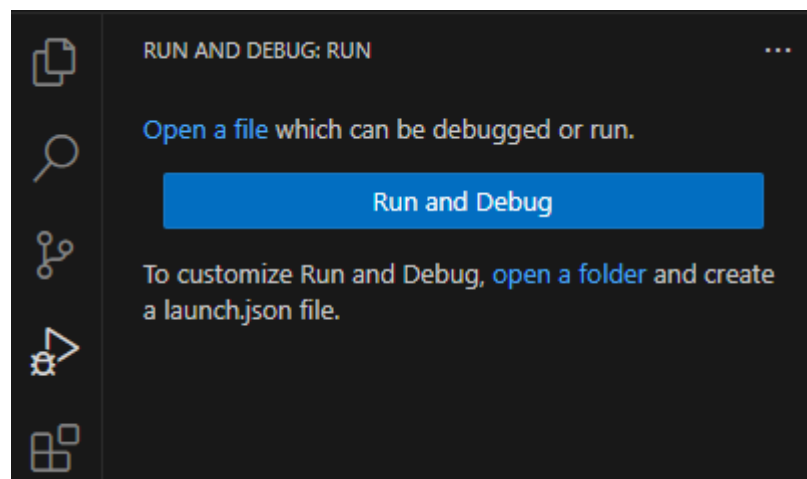


Рисунок 22 – Вкладка Run and Debug

Вкладка Extensions – это магазин дополнений, которые расширяют функционал VS Code. Она имеет функции нахождения и добавления плагинов для языков программирования, тем оформления и инструментов, позволяет включать/отключать или удалять расширения. Данный магазин с расширениями показан на рисунке 23.

Полезные расширения для создания мини-приложения в Telegram: Python (поддержка синтаксиса и отладки), REST client (тестирование API), GitLens (расширенные возможности Git).

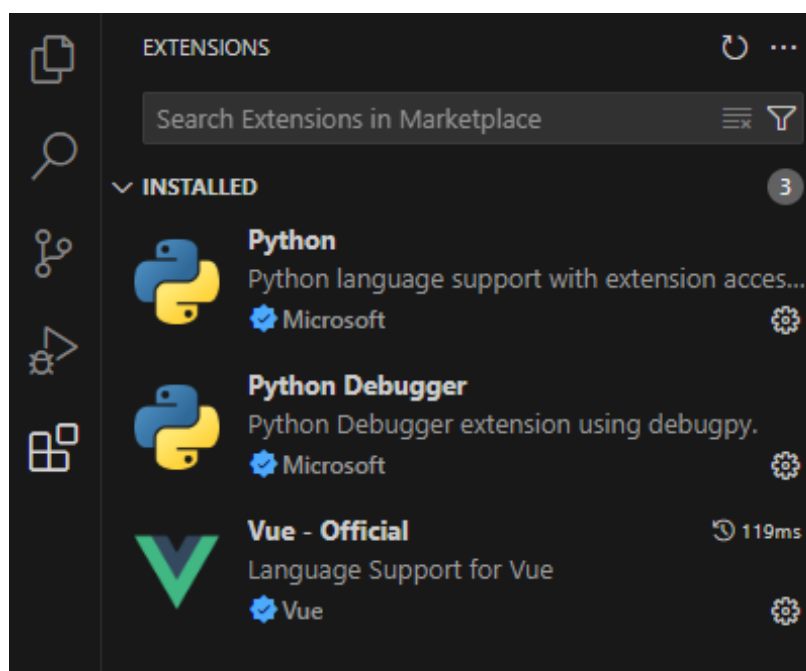


Рисунок 23 – Вкладка Extensions

В более поздних версиях VS Code появилась вкладка Testing, и она предназначена для работы с тестами. Также позволяет автоматически находить тесты в проекте (например, pytest для Python), выполнять отдельные тесты или группы, показывает результаты и ошибки.

Таким образом, использование этих инструментов в комплексе значительно ускоряет процесс создания и поддержки бота, а в дальнейшем и мини-приложения, делая разработку более удобной и эффективной.

1.2 Проектирование и разработка

1.2.1 Разработка backend-части

В процессе разработки backend-части мини-приложения для мониторинга криптовалютных активов в Telegram был выбран стек технологий на Python, включающий FastAPI для создания API, SQLAlchemy для работы с базой данных и APScheduler для управления фоновыми задачами. Начало работы над проектом заключалось в настройке окружения и установке необходимых зависимостей, перечисленных в файле requirements.txt. Ключевыми зависимостями стали fastapi для создания веб-сервера, uvicorn в качестве ASGI-сервера, SQLAlchemy для ORM, psycopg2-binary для подключения к PostgreSQL, а также библиотеки aiohttp для HTTP-запросов и python-dotenv для управления переменными окружения [6]. Были выбраны следующие версии зависимостей и библиотек:

```
fastapi==0.115.12
uvicorn==0.34.2
SQLAlchemy==2.0.41
psycopg2-binary==2.9.10
aiohttp==3.12.2
python-dotenv==1.1.0
```

Инициализация FastAPI-приложения была выполнена в файле main.py, где также настроены CORS для беспрепятственного взаимодействия с фронтендом.

```

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

```

Здесь же был реализован механизм фоновых задач с использованием APScheduler. Планировщик задач был настроен на выполнение двух ключевых функций: синхронизацию о ценах каждые 50 секунд через функцию `sync_price` и отправку уведомлений пользователям каждую минуту с помощью `notify_users`. Запуск планировщика происходит автоматически при старте приложения благодаря декоратору `@app.on_event("startup")`.

```

@app.on_event("startup")
async def startup_event():
    print("✅ FastAPI запущен и планировщик работает")
    scheduler.add_job(
        notify_users,
        trigger=IntervalTrigger(minutes=1),
        id="notify_users",
        name="Check and notify about coin prices",
        replace_existing=True
    )
    scheduler.add_job(
        sync_price,
        trigger=IntervalTrigger(seconds=50),
        id="sync_price",
        name="Sync prices",
        replace_existing=True
    )
    scheduler.start()

```

Для работы с внешним API криптовалют используется сервис CoinGecko [8]. Основными критериями выбора стали бесплатный доступ или разумные тарифные планы, возможность частых запросов (не менее одного в минуту), актуальность данных с минимальной задержкой и простота интеграции с Python.

Для того чтобы получить API-ключ для отображения динамических цен, конвертации и отправки уведомлений пользователям необходимо зарегистрироваться и создать этот ключ. На рисунке 24 показана главная страница сайта CoinGecko.

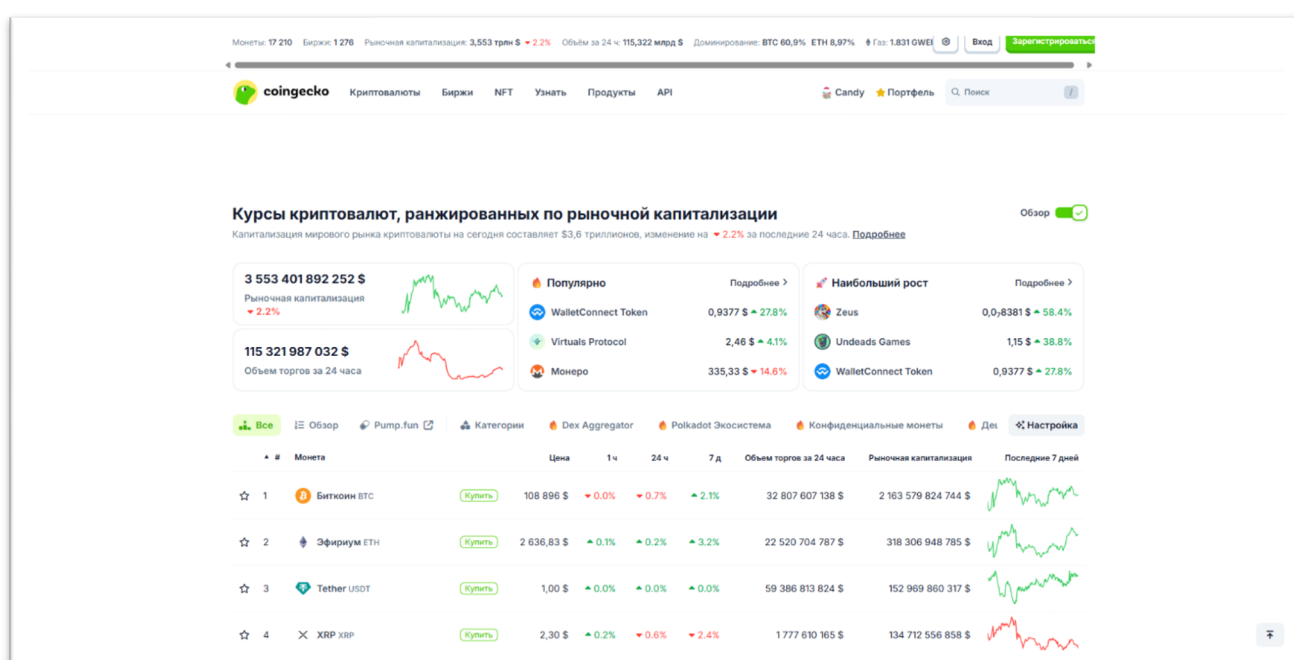


Рисунок 24 – Главная страница сайта CoinGecko

После осуществляется переход во вкладку «Войти» и регистрация с помощью email, как показано на рисунке 25.

Далее подтверждается код на почте и происходит успешная авторизация на сайте для перехода в девелоперский дашборд, чтобы создать необходимый API-ключ, как показано на рисунках 26 и 27.

×

Добро пожаловать на CoinGecko

Войдите или зарегистрируйтесь за считанные секунды


Продолжить с Google


Продолжить с Apple

или

Продолжить с email

Продолжая, вы подтверждаете, что прочитали и полностью принимаете наши [Условия обслуживания](#) и [Политика конфиденциальности](#).

Рисунок 25 – Регистрация на CoinGecko

Мой аккаунт

Логин и безопасность

Перейти на Премиум

Для разработчиков

Девелоперский дашборд

Выйти

Рисунок 26 – Контекстное меню личного кабинета

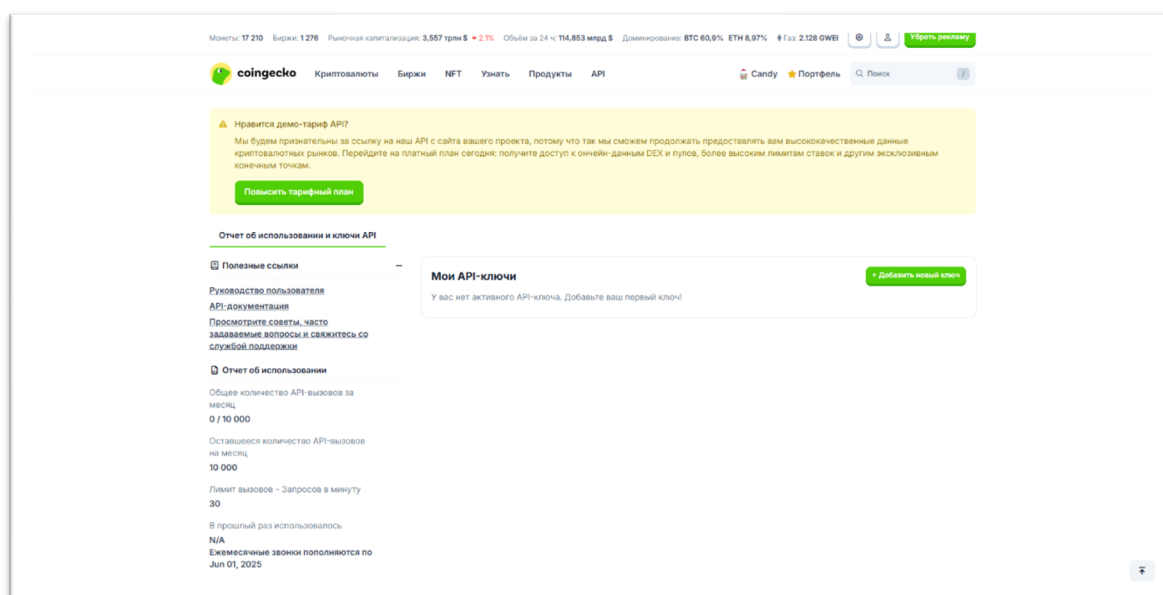


Рисунок 27 – Девелоперский дашборд для создания API-ключа

Далее всё становится интуитивно понятно – осуществляется переход с помощью кнопки «Добавить новый ключ» и там необходимо ввести метку API (вводится произвольное название). После этого успешно создаётся API-ключ, как показано на рисунке 28.

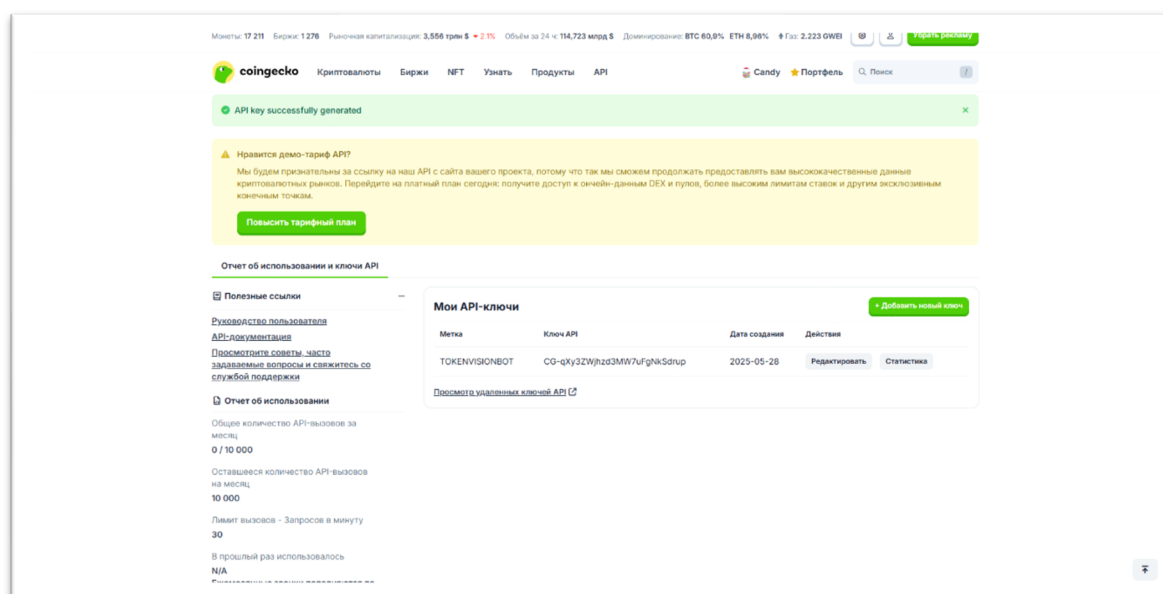


Рисунок 28 – Получение API-ключа для мониторинга и анализа цен на криптовалюту

После успешного создания необходимого ключа, на почту приходит сообщение, как показано на рисунке 29, с документацией о том, как описывать его в коде для получения достоверных данных о криптовалюте.

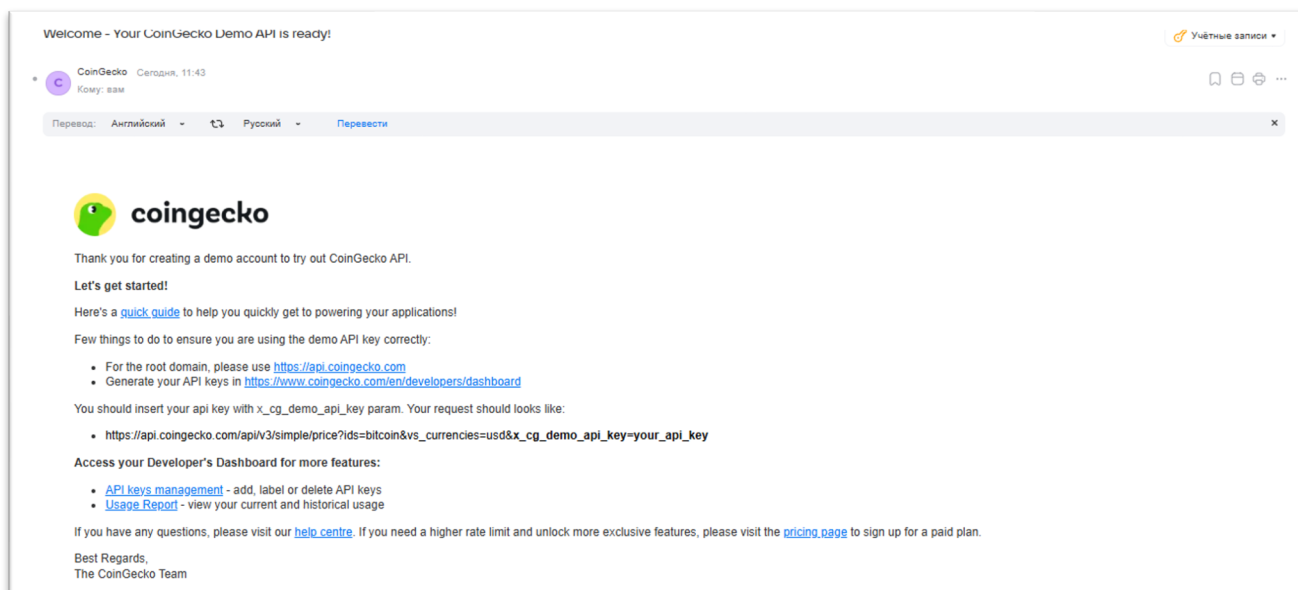


Рисунок 29 – Сообщение от CoinGecko после создание API-ключа

Из сообщения понятно, что ссылка на запросы в коде выглядит так: `https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd&C` `G-qXy3ZWjhzd3MW7uFgNkSdrup` (Получение актуальной цены на BTC в USD) [8]. После проделанных действий происходит применение полученного API-ключа, как описано ниже.

В модуле `services.py` была реализована асинхронная функция `fetch_crypto_data`, которая отправляет запрос к CoinGecko API и получает актуальные данные о ценах.

```
async def fetch_crypto_data():
    params = {
        'vs_currency': 'usd',
        'ids': ','.join(c['id'] for c in crypto_list),
        'price_change_percentage': '24h',
        'x_cg_demo_api_key': API_KEY
    }
```

Полученные данные затем сохраняются в базу данных PostgreSQL через функцию `save_prices`, где каждая запись содержит идентификатор монеты, её текущую цену и временную метку. Для обработки ошибок при работе с базой данных использовалась конструкция `try-except-finally`, гарантирующая корректное закрытие сессии даже в случае возникновения исключений.

```
def save_prices(data: list[dict]):
    db: Session = SessionLocal()
    try:
        for coin in data:
            result = TrackingResult(
                coin_id=coin['id'],
                price=coin['current_price'],
                timestamp=datetime.utcnow()
            )
            db.add(result)
        db.commit()
        print("✅ Цены сохранены в БД")
    except Exception as e:
        print(f"❌ Ошибка сохранения в БД: {e}")
        db.rollback()
    finally:
        db.close()
```

Модели данных, описанные в `models.py`, включают две основные таблицы: `TrackingRuleDB` для хранения правил отслеживания, установленных пользователями, и `TrackingResult` для хранения исторических данных о ценах. В `TrackingRuleDB` сохраняется `chat_id` пользователя, период уведомлений (минуты, часы или дни) и список отслеживаемых криптовалют. Модель `TrackingResult` содержит идентификатор монеты, её цену и временную метку.

Для подключения к базе данных используется строка подключения, формируемая на основе переменного окружения, что обеспечивает гибкость при развертывании в разных средах [8].

```
class TrackingRuleDB(Base):
    __tablename__ = "tracking_rules"
    chat_id = Column(Integer, primary_key=True, index=True)
    period = Column(String(1), nullable=False) # 'm', 'h', 'd'
    coins = Column(ARRAY(String), nullable=False)
    last_notified_at = Column(DateTime, nullable=True)

class TrackingResult(Base):
    __tablename__ = "tracking_results"
    id = Column(Integer, primary_key=True, index=True)
    coin_id = Column(String, nullable=False)
    price = Column(Float, nullable=False)
    timestamp = Column(DateTime, default=datetime.utcnow)
```

Логика уведомлений, реализованная в `notifier.py`, проверяет условия для отправки сообщений пользователям. Для каждого правила отслеживания вычисляется время, прошедшее с последнего уведомления, и если оно превышает установленный пользователем период, формируется и отправляется сообщение с актуальной информацией о ценах. Отправка сообщений осуществляется через Telegram Bot API с использованием функции `send_message` из модуля `tg.py`. Эта функция принимает `chat_id` пользователя и текст сообщения, затем отправляет запрос к API Telegram для доставки сообщения.

```
await send_message(str(rule.chat_id), msg)
```

Интеграция с Telegram реализована через официальный Bot API [8]. В файле `tg.py` описан простой клиент для отправки сообщений, использующий `aiohttp` для асинхронных HTTP-запросов [8]. Ключ доступа к боту хранится в переменных окружения, что обеспечивает безопасность и возможность

легкой смены токена при необходимости.

```
async with aiohttp.ClientSession() as session:
    async with session.post(url, json=payload) as response:
        if response.status != 200:
            error = await response.text()
            print(f"❌ Ошибка отправки сообщения: {error}")
        else:
            print(f"✅ Сообщение отправлено в {chat_id}")
```

Для валидации данных, поступающих от пользователей через API, используются модели Pydantic, описанные в `dto.py`. Например, модель `TrackingRule` строго определяет структуру данных для создания правил отслеживания, включая тип периода, идентификатор чата и список монет.

```
class TrackingRule(BaseModel):
    period: Literal['m', 'h', 'd']
    chatId: int
    coins: List[str]
```

Развертывание backend-части организовано с использованием Docker и docker-compose. В Dockerfile указаны инструкции для сборки образа Python с установленными зависимостями, а в docker-compose.yml описаны сервисы для запуска backend и базы данных PostgreSQL. Такое решение обеспечивает простоту развертывания и масштабируемость приложения. Для обработки входящих HTTP-запросов и балансировки нагрузки настроен Nginx, конфигурация которого хранится в файле `nginx.conf` [8].

В результате backend-часть приложения получила все необходимые функции для работы: сбор данных о криптовалютах, их хранение в базе данных, обработку правил отслеживания и отправку уведомлений через Telegram.

1.2.2 Разработка frontend-части

Следующим этапом разработки стала реализация фронтенд-части на Vue.js, которая предоставляет пользователям удобный интерфейс для

управления своими правилами отслеживания и просмотра актуальной информации о криптовалютах.

Фронтенд-часть мини-приложения для мониторинга криптовалютных активов реализована с использованием фреймворка Vue.js 3, что позволило создать реактивный и производительный интерфейс, оптимизированный для работы внутри Telegram WebApp [8]. Архитектура проекта построена по принципу компонентного подхода, где каждый функциональный элемент представляет собой независимый компонент с собственной логикой и стилями.

Основной точкой входа во фронтенд-приложение является файл main.js, где происходит инициализация Vue-приложения. Здесь же реализована интеграция с Telegram WebApp API, что позволяет приложению корректно работать внутри Telegram-клиентов. При обнаружении объекта window.Telegram.WebApp выполняется его инициализация методом ready() и расширение интерфейса методом expand() [8]. Для удобства разработки вне Telegram-окружения добавлена заглушка с тестовым ID пользователя.

```
if (window.Telegram?.WebApp) {  
  try {  
    window.Telegram.WebApp.ready()  
    window.Telegram.WebApp.expand()  
    if (!window.Telegram.WebApp.initDataUnsafe?.user) {  
      window.Telegram.WebApp.initDataUnsafe = {  
        user: { id: 123456789 }  
      }  
    }  
  } catch (e) {  
    console.error('Telegram WebApp init error:', e)  
  }  
}
```

Ядро приложения сосредоточено в компоненте App.vue, который выполняет роль контейнера для всех представлений. Здесь реализована анимация перехода между страницами через компонент transition и система маршрутизации Vue Router. Важной особенностью является автоматическое сохранение идентификатора пользователя Telegram в localStorage при монтировании компонента, что обеспечивает сохранность пользовательских данных между сеансами.

```
<template>
<div class="app-container">
  <router-view v-slot="{ Component }">
    <transition name="fade" mode="out-in">
      <component :is="Component"/>
    </transition>
  </router-view>
  <Navbar/>
</div>
</template>
```

Навигация между разделами приложения реализована через компонент Navbar.vue, который динамически отображает доступные маршруты из конфигурации роутера. Каждая навигационная ссылка включает иконку и текст, взятые из мета-полей соответствующих маршрутов. Активное состояние ссылки визуально выделяется изменением цвета и фона.

```
const routes = computed(() =>
  router.getRoutes().filter(r => r.meta.icon)
)
```

Основной функционал мониторинга криптовалют сосредоточен в компоненте HomeView.vue, который отображает список криптовалютных активов с их текущими ценами и изменением стоимости. Данные получаются через CoinGecko API с использованием библиотеки axios [8].

Для обеспечения актуальности информации реализован механизм периодического обновления данных через `setInterval` с интервалом в 60 секунд.

```
const fetchData = async () => {
  try {
    const response = await axios.get(API_URL, {
      params: {
        vs_currency: 'usd',
        ids: cryptoList.value.map(c => c.id).join(','),
        price_change_percentage: '24h'
      }
    })
    cryptoList.value = cryptoList.value.map(coin => {
      const updatedData = response.data.find(c => c.id === coin.id) || {}
      return {
        ...coin,
        current_price: updatedData.current_price || 0,
        price_change_percentage_24h:
          updatedData.price_change_percentage_24h || 0
      }
    })
  } catch (err) {
    error.value = 'Не удалось загрузить данные'
  }
}
```

Каждая криптовалюта отображается в виде карточки через компонент `CryptoCard.vue`, который принимает объект с данными о криптовалюте через `props`. Карточка показывает символ и название криптовалюты, текущую цену с форматированием для локали и процент изменения цены с цветовой индикацией (зеленый для роста, красный для падения). Для изображений криптовалют реализована обработка ошибок загрузки с отображением fallback-контента [8].

```

<span class="price-change"
:class="{ positive:crypto.price_change_percentage_24h >= 0,
negative: crypto.price_change_percentage_24h < 0 }">
{{ crypto.price_change_percentage_24h >= 0 ? '↑' : '↓' }}
{{Math.abs(crypto.price_change_percentage_24h).toFixed(2)}}%
</span>

```

Функционал конвертации валют реализован в компоненте ConverterView.vue, который позволяет пользователям пересчитывать стоимости между различными криптовалютами и цифровым долларом США (USDT). Компонент включает два выпадающих списка для выбора исходной и целевой валюты, поле для ввода суммы и кнопку для смены валют местами. Курсы конвертации рассчитываются на основе данных, получаемых от CoinGecko API [8].

```

const convertCurrency = () => {
const rate=rates.value[fromCurrency.value][toCurrency.value]
conversionRate.value = formatNumber(rate)
convertedAmount.value = formatNumber(amount.value * rate)
}

```

Настройки уведомлений вынесены в отдельный компонент NotificationsView.vue, где пользователи могут выбирать криптовалюты для отслеживания и устанавливать частоту уведомлений. Состояние переключателей синхронизируется с бэкендом через REST API, сохраняя предпочтения пользователя на сервере. При сохранении настроек выполняется POST-запрос к эндпоинту /track бэкенда.

```

const saveSettings = async () => {
try {
await fetch(`${domain}/track`, {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({
period: frequency.value,

```

```

chatId: chatId.value,
coins: trackedCoins.value
}))
}))
message.value = 'Настройки успешно сохранены!'
} catch (error) {
message.value = 'Произошла ошибка: ' + error.message
}
}

```

Маршрутизация между компонентами организована через Vue Router с историей HTML5. Конфигурация роутера (index.js) определяет три основных маршрута: главную страницу с курсами валют, конвертер и настройки уведомлений. Каждый маршрут содержит мета-данные для отображения в навигационной панели.

```

const routes = [
{
path: '/',
name: 'home',
component: HomeView,
meta: { title: 'Курсы валют', icon: '🏠' }
}
]

```

Сборка проекта настроена через Vite (vite.config.js), что обеспечивает быструю разработку с горячей перезагрузкой модулей и оптимизированную production-сборку. Конфигурация включает алиасы для удобного импорта компонентов, настройки dev-сервера и подключение плагина Vue DevTools для отладки.

```

export default defineConfig({
plugins: [vue(), vueDevTools()],
resolve: {
alias: {

```

```
'@': fileURLToPath(new URL('./src', import.meta.url))
}
}
})
```

Стили приложения реализованы с использованием CSS-переменных для цветовых тем, что обеспечивает легкую поддержку темного/светлого режимов. Адаптивная верстка гарантирует корректное отображение на различных устройствах. Все компоненты используют scoped-стили для изоляции CSS-правил и предотвращения конфликтов.

Таким образом, фронтенд-часть приложения представляет собой полнофункциональный интерфейс для мониторинга криптовалютных активов, конвертации валют и управления уведомлениями, оптимизированный для работы внутри Telegram и отвечающий современным стандартам веб-разработки.

1.2.3 Создание мини-приложения в Telegram

Для того чтобы сделать своё мини-приложение в мессенджере Telegram, потребуется создать своего бота с помощью бота-помощника @BotFather. Для этого необходимо ввести в общем поиске название бота, далее зайти в него и начать работу с ним с помощью кнопки /start, как показано на рисунке 30.

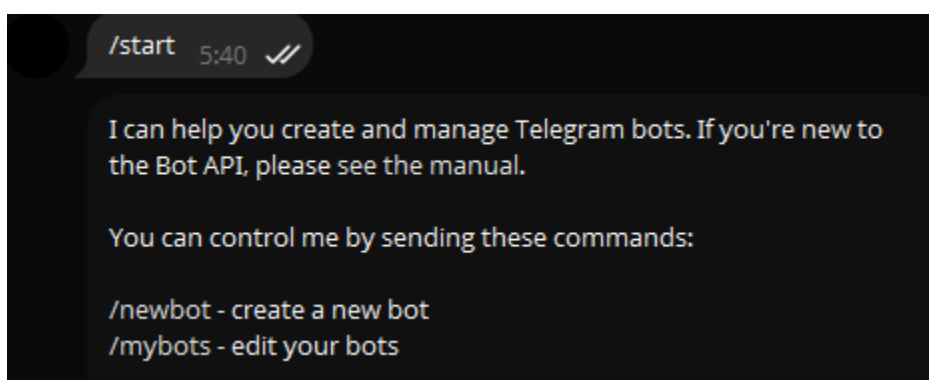


Рисунок 30 – Запуск бота-помощника BotFather

Далее необходимо выбрать команду /newbot для создания своего бота, как показано на рисунке 31.

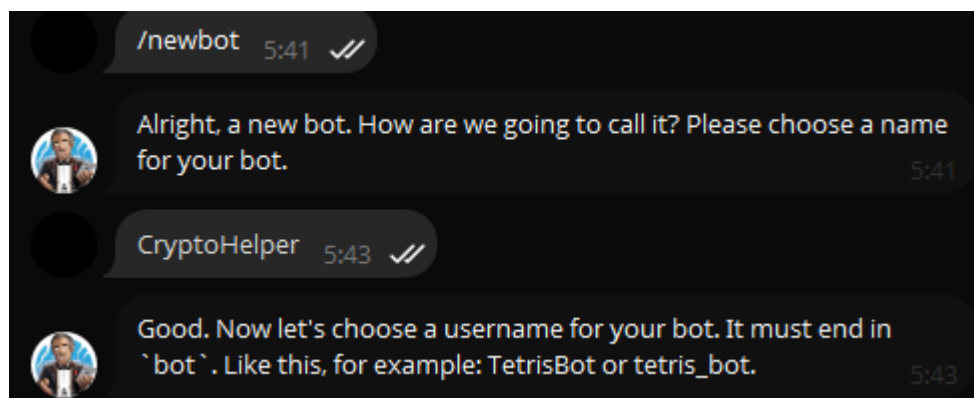


Рисунок 31 – Создание нового бота

Далее бот-помощник просит выбрать имя пользователя в Telegram для создаваемого бота. После выбора такого имени, которое ещё не занято, бот успешно формируется и BotFather предоставляет токен для доступа к HTTP API, как показано на рисунке 32.

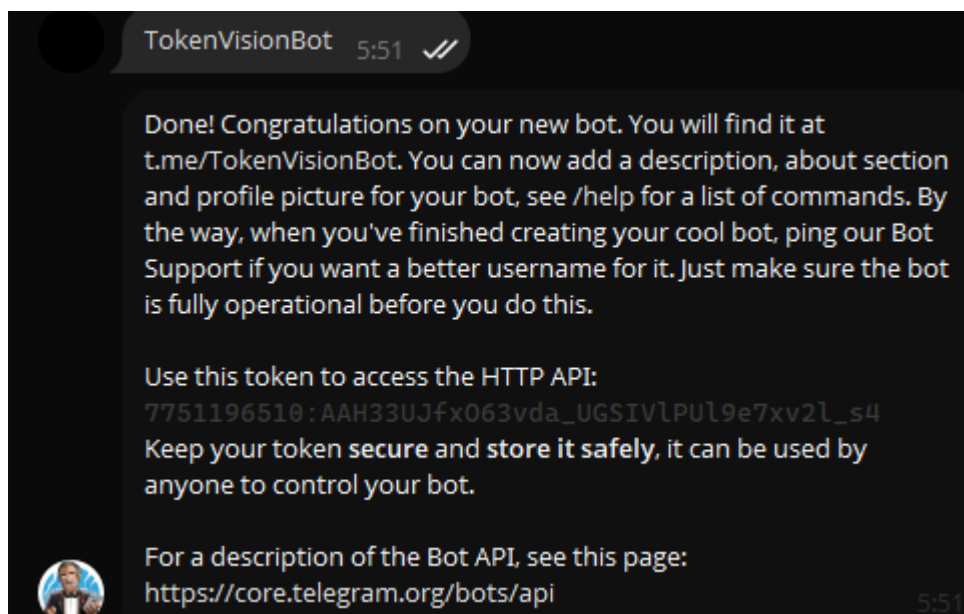


Рисунок 32 – Выбор имени пользователя

После успешного создания собственного бота, необходимо сделать конфигурацию в мини-приложение. Для этого достаточно перейти в пункт настроек этого бота в контекстном меню чата с ботом-помощником и выбрать пункт «Configure Mini App», как показано на рисунке 33.

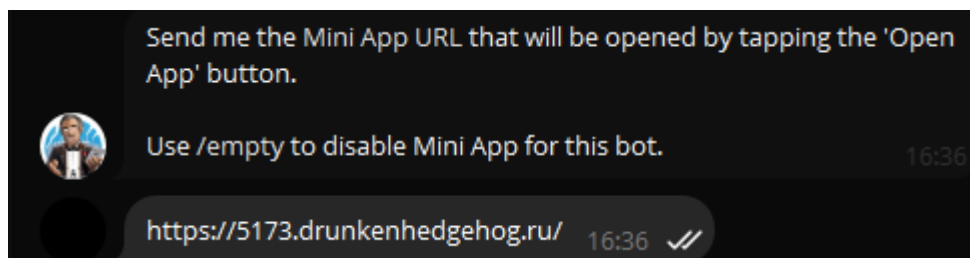


Рисунок 33 – Конфигурация бота в мини-приложение

1.3 Анализ результатов

Вход в получившееся мини-приложение может быть осуществлён в мобильной версии Telegram, с помощью Telegram Desktop, а также через браузер. После успешной авторизации через любой из способов пользователю предоставляется простой и понятный интерфейс. На главной странице, которая относится ко вкладке «Курсы валют» пользователь может увидеть актуальные цены на самые популярные криптовалютные токены, такие как Bitcoin, Ethereum, USD Tether, Solana, Tron, как показано на рисунках 34-36.

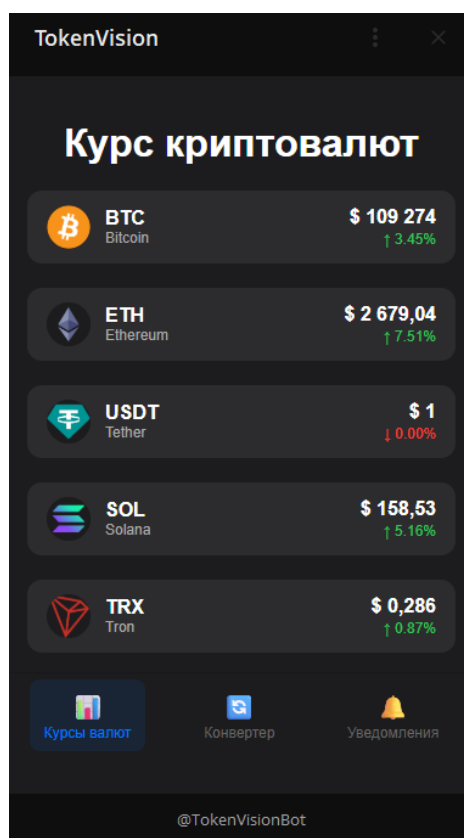


Рисунок 34 – Вход через Telegram Desktop

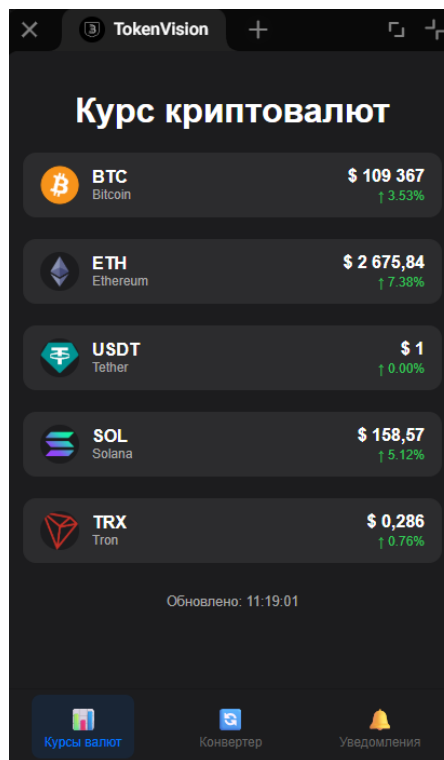


Рисунок 35 – Вход через браузер

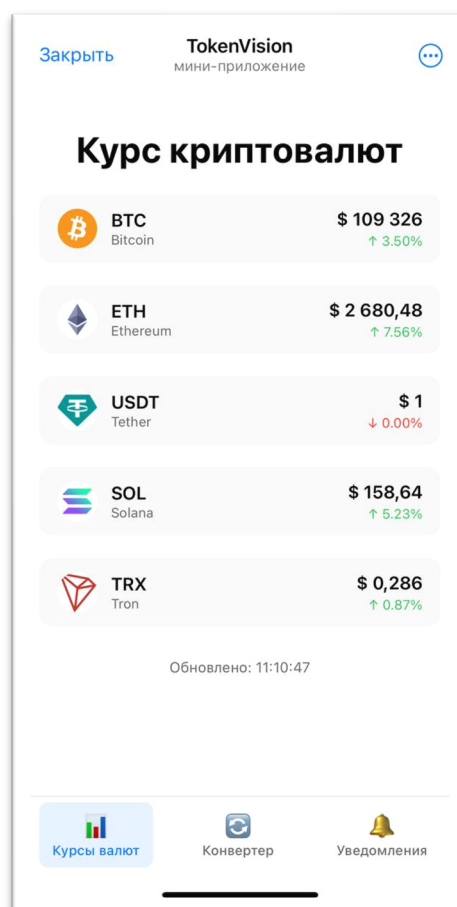


Рисунок 36 – Вход через мобильную версию

На второй вкладке на нижней панели мини-приложения, которая называется «Конвертер», пользователю предлагается возможность конвертировать одну криптовалюту в другую и наоборот, используя выпадающие списки из предлагаемых токенов справа от полей вводимой и получаемой монет. Также внизу окна конвертации указывается цена за одну единицу монеты на данный момент. Эта цена меняется динамически каждую минуту, также как и на главной странице приложения. Ниже, на рисунках 37-38, показаны два примера конвертации различных монет.

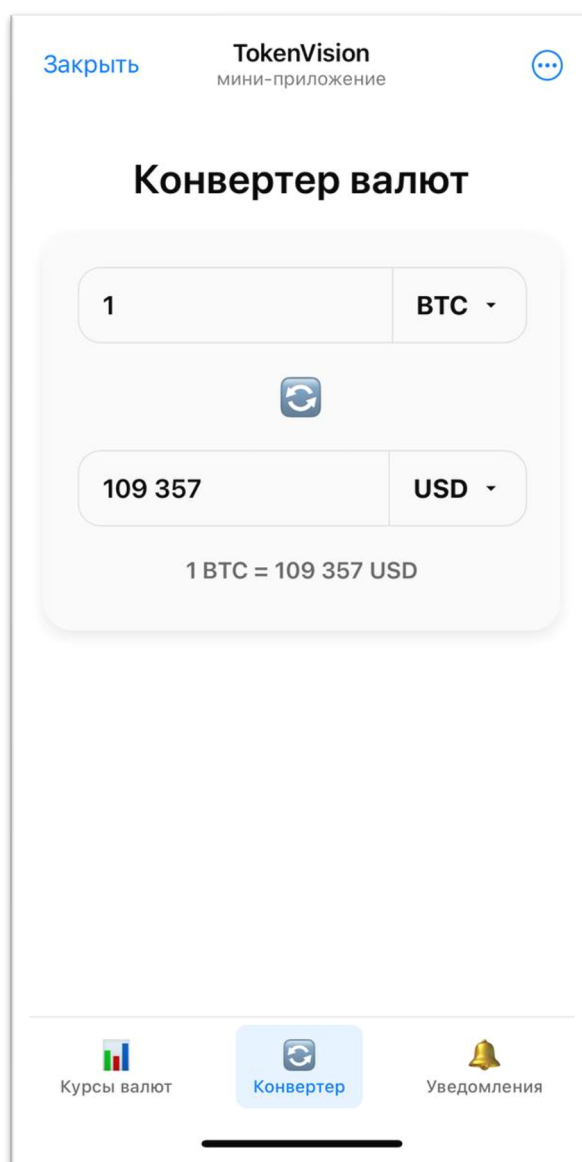


Рисунок 37 – Конвертация BTC в USD

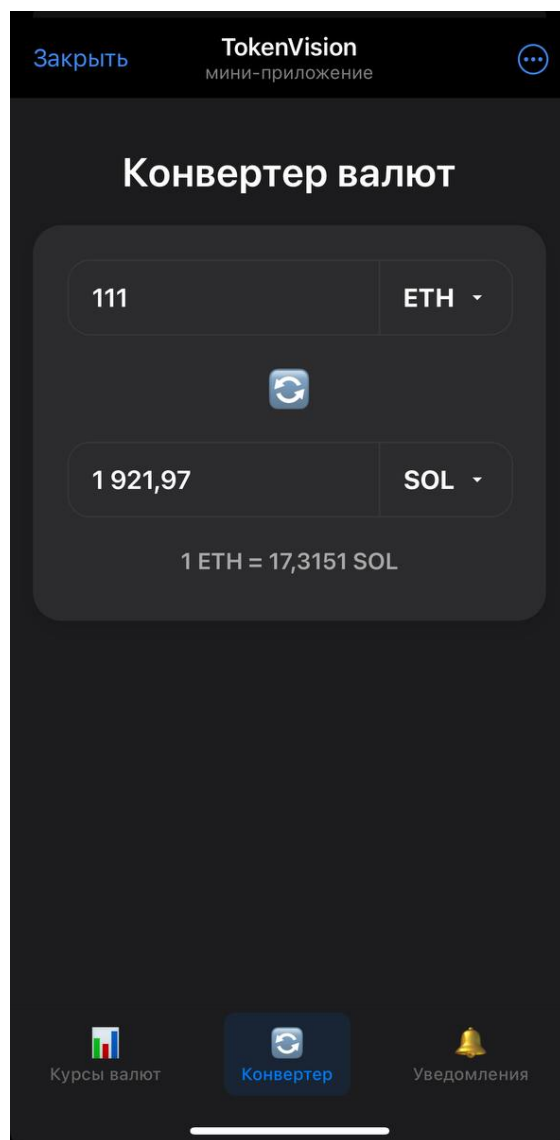


Рисунок 38 – Конвертация ETH в SOL

Мини-приложение умеет подстраиваться под текущую тему Telegram – будь то светлый или темный режим. Это сделано не только для визуальной составляющей: автоматическая смена цветовой гаммы действительно помогает глазам меньше уставать, особенно когда пользуешься приложением вечером или ночью. Все цвета интерфейса меняются плавно и моментально, без каких-то видимых перезагрузок или мерцаний экрана. Такой подход к оформлению показывает, что задумка заключается не только в функциональности, но в комфорте пользователей в разных условиях освещения.

На последней вкладке получившего мини-приложения под названием «Уведомления», пользователь может настроить получение сообщений от бота о изменении цен на предпочитаемые монеты криптовалюты за желаемый период времени. Пользователь может получать уведомления каждую минуту, каждый час или раз в день, в зависимости от сделанного выбора, как показано на рисунках 39-40.

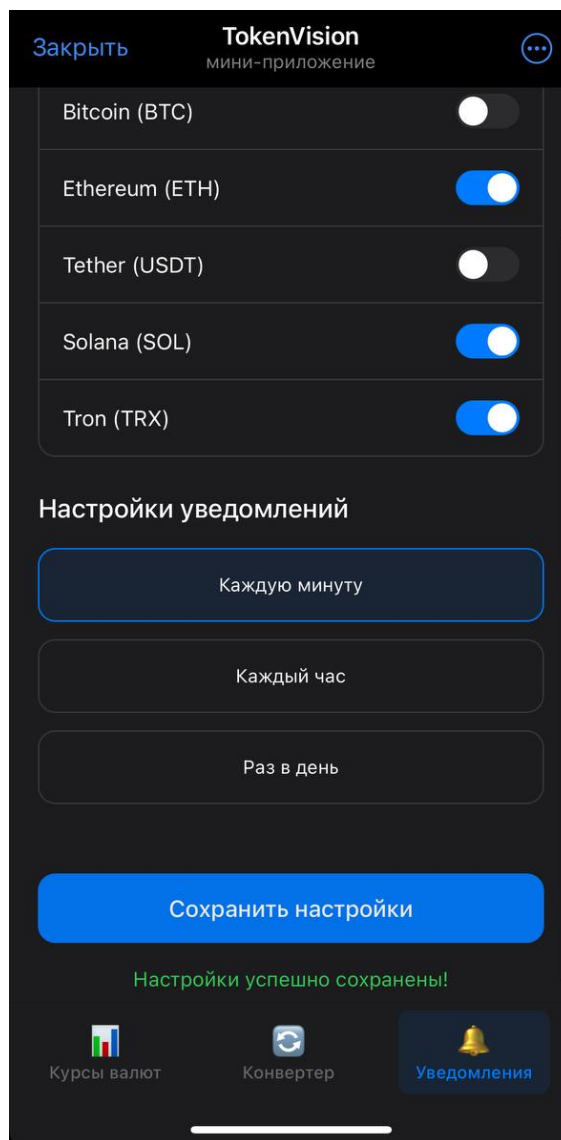


Рисунок 39 – Настройка уведомления для ETH, SOL, TRX



Рисунок 40 – Уведомления от бота с wybranными монетами

Таким образом, был произведён анализ полученного мини-приложения в Telegram, которое соответствует первоначальной идее. Было реализовано простое и в тоже время удобное средство для мониторинга, конвертации и уведомлений об изменении цен за определённый период времени. Минималистичный интерфейс и лёгкое взаимодействие с функционалом мини-приложения позволяет новичкам в сфере крипторынка быстро и понятно отслеживать динамику роста и падения криптовалют в целях заработка и инвестиций.

2 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда при разработке программного обеспечения на персональном компьютере, установке и эксплуатации мини-приложения, а также способы минимизации пагубных факторов на здоровье человека.

2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения

Длительная работа, в сидячем положении за компьютером оказывает негативное влияние на здоровье человека. Воздействие неблагоприятных факторов приводит к снижению работоспособности, вызываемому утомлением. Длительное нахождение оператора в зоне комбинированного воздействия различных неблагоприятных факторов может привести к профессиональному заболеванию.

Чтобы минимизировать вредное влияние компьютера, необходимо следовать следующим правилам:

- отрегулируйте высоту кресла или подберите стул, чтобы при сидении спина была ровной, а предплечья лежали на столе;
- экран должен находиться на уровне глаз, на расстоянии вытянутой руки;
- избегайте бликов и теней, используйте настольную лампу при необходимости;
- каждые 45-60 минут вставайте из-за стола, разминайтесь, делайте упражнения для глаз; старайтесь не проводить за компьютером более 6-8 часов в день;
- держите спину ровной, не сутультесь;
- не игнорируйте правила электробезопасности – используйте заземленные розетки, не перегружайте сеть;

– соблюдайте правила пожарной безопасности, держите на рабочем огнетушитель, не оставляйте без присмотра включенные электроприборы.

2.1.1 Оборудование рабочего места

Ниже представлены основные требования, которым должно удовлетворять рабочее место [7]:

- расстояние от глаз до экрана должно быть в пределах 600-700 мм;
- монитор должен располагаться ниже уровня глаз. Угол между направлением при взгляде прямо и направлением на центр экрана должен составлять 10-15 градусов;
- рабочий стол должен регулироваться по высоте в пределах 680-800 мм, при отсутствии такой возможности его высота должна составлять 725 мм;
- клавиатуру следует располагать на поверхности стола на расстоянии 100-300 мм от края, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности, отделенной от основной столешницы;
- рабочий стол должен иметь пространство для ног высотой не менее 600 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм [7].

На рисунке 41 изображена организация рабочего места, которая отвечает указанным требованиям.

Рабочее место с дисплеем должно быть эргономичным, то есть максимально удобным и не наносящим вреда здоровью.

При правильно организованном рабочем месте должна обеспечиваться правильная посадка пользователя ПЭВМ, изображённая на рисунке 42.



Рисунок 41 – Организация рабочего места

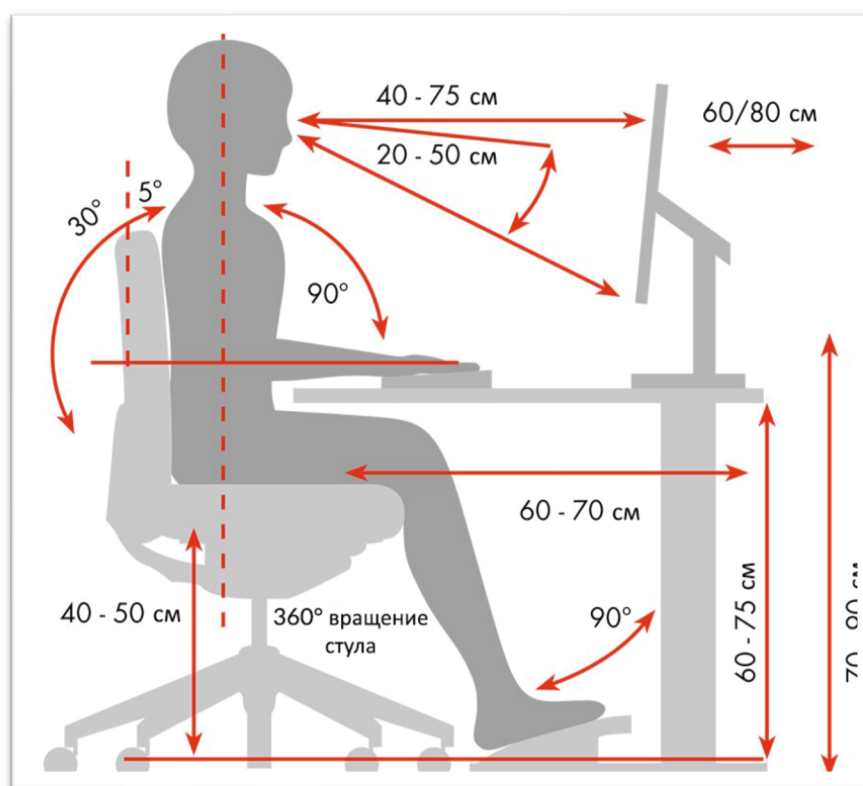


Рисунок 42 – Правильная посадка за ПК

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.18.001 ПЗ

Лист

59

Посадка должна обеспечивать выполнение следующих условий:

- запястья должны находиться в прямом положении;
- плечи находятся в расслабленном состоянии;
- локти должны полностью лежать на столе, не свисая и располагаться

перпендикулярно полу (угол в локтевых суставах должен составлять 90-100 градусов).

2.1.2 Освещенность рабочего места

Нормы освещенности устанавливаются с учетом характера выполняемой работы, места и поверхности. Для ЭВМ и дисплеев – комбинированное освещение не менее 400 люкс, а общее освещение не менее 200 люкс [9].

На рабочем месте необходимо обеспечивать большую равномерность яркости, исключая наличие ярких и блестящих предметов, для снижения монотонности в поле зрения рекомендуется отдельные пестрые поверхности.

Для освещения используются в основном потолочные или встроенные светильники с люминесцентными лампами. Для исключения засвечивания экранов прямыми световыми потоками светильники общего освещения располагают сбоку от рабочего места, параллельно линии зрения оператора [9].

Заключение

Мини-приложение в Telegram для мониторинга и анализа криптовалютных активов представляет собой современное и удобное решение, отвечающее потребностям трейдеров и инвесторов. Объединяя преимущества мессенджера (доступность, социальные функции) с мощными аналитическими инструментами, такие сервисы могут стать новым стандартом для работы с крипторынком.

Развитие экосистемы Telegram Open Network (TON) и встроенных финансовых сервисов открывает дополнительные возможности для интеграции криптоплатежей и децентрализованных приложений (DApps). В перспективе подобные мини-приложения могут не только отображать данные, но и позволять совершать сделки, управлять портфелем и взаимодействовать с DeFi-протоколами прямо в мессенджере.

Таким образом, создание мини-приложения для анализа криптовалют не только актуальная задача в рамках текущих трендов, но и перспективное направление для дальнейшего развития в социальных и коммуникационных платформах.

					<i>ПГУ 1.090301.18.001 ПЗ</i>	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Telegram Mini App. Как создать Web App с нуля [Электронный ресурс] / Режим доступа: <https://habr.com/ru/companies/amvera/articles/838180/> свободный (дата обращения: 14.05.2025).
2. Общий обзор Telegram Mini App: особенности, преимущества, ограничения и примеры [Электронный ресурс] / Режим доступа: <https://habr.com/ru/articles/886864/> свободный (дата обращения: 19.04.2025).
3. Wikipedia. [Электронный ресурс] / Режим доступа: <https://ru.wikipedia.org/wiki/Telegram> свободный (дата обращения: 29.04.2025).
4. Telegram Mini Apps. [Электронный ресурс] / Режим доступа: <https://core.telegram.org/bots/webapps> свободный (дата обращения: 30.04.2025).
5. Создание переменных для хранения токена бота и HTTP-запросов. [Электронный ресурс] / Режим доступа: <https://skillbox.ru/media/code/telegram-bot-api/> свободный (дата обращения: 30.04.2025).
6. Справочник по Bot API. [Электронный ресурс] / Режим доступа: <https://tlgrm.ru/docs/bots/api> свободный (дата обращения: 17.05.2025).
7. Требования по обустройству помещений и рабочих мест, предназначенных для работы на компьютере и прочей офисной технике [Электронный ресурс] / Режим доступа: https://www.consultant.ru/document/cons_doc_LAW_18895/df3eaac6ab127894002172933f9cd660e61df962/ свободный (дата обращения: 20.05.2025).
8. Документация по получению API-ключа для динамического изменения цен на криптовалюту [Электронный ресурс] / Режим доступа: <https://docs.coingecko.com/v3.0.1/reference/introduction> свободный (дата обращения: 26.05.2025).

9. ГОСТ IEC 61131-2-2012 Часть 2 Требования к оборудованию и
испытания [Электронный ресурс] / Режим доступа:
<https://www.reglament.by/wp-content/uploads/docs/gost/gost-iek-61131-2-2012.pdf>
свободный (дата обращения: 10.05.2025).

					<i>ПГУ 1.090301.18.001 ПЗ</i>	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

Листинги backend-части

Приложение А.1

Файл «main.py»

```
import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.router import router
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from apscheduler.triggers.interval import IntervalTrigger

from app.services import sync_price
from notifier import notify_users

app = FastAPI()

# Разрешаем все CORS-запросы
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

scheduler = AsyncIOScheduler()

@app.on_event("startup")
async def startup_event():
    print("✅ FastAPI запущен и планировщик работает")
```

					ПГУ 1.090301.18.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

scheduler.add_job(
    notify_users,
    trigger=IntervalTrigger(minutes=1),
    id="notify_users",
    name="Check and notify about coin prices",
    replace_existing=True
)

scheduler.add_job(
    sync_price,
    trigger=IntervalTrigger(seconds=50),
    id="sync_price",
    name="Sync prices",
    replace_existing=True
)

scheduler.start()

app.include_router(router)

if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=4157,
reload=True)

```

Приложение А.2

Файл «router.py»

```

from fastapi import APIRouter, HTTPException

from app import tg
from app.models import get_tracking_rule, save_tracking_rule
from app.dto import TrackingRule

router = APIRouter()

@router.post("/track")
async def create_tracking_rule(rule: TrackingRule):

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

# await tg.send_message(str(rule.chatId), "...")
    save_tracking_rule(rule.chatId, rule.period, rule.coins)
    return {"message": "Tracking rule saved", "data": rule}

@router.get("/track/{chat_id}")
async def get_tracking_rule_endpoint(chat_id: int):
    rule = get_tracking_rule(chat_id)
    if not rule:
        raise HTTPException(status_code=404, detail="Rule not
found")
    return {
        "chatId": rule.chat_id,
        "period": rule.period,
        "coins": rule.coins,
    }

```

Приложение А.3

Файл «services.py»

```

import os
import aiohttp
import asyncio
import dotenv
from datetime import datetime

from sqlalchemy.orm import Session
from app.models import SessionLocal, TrackingResult

dotenv.load_dotenv()

API_URL = 'https://api.coingecko.com/api/v3/coins/markets'
API_KEY = os.getenv('COINGECKO_TOKEN')

crypto_list = [
    {"id": "bitcoin"},

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

```

{"id": "ethereum"},
    {"id": "tether"},
    {"id": "solana"},
    {"id": "tron"}
]

async def fetch_crypto_data():
    params = {
        'vs_currency': 'usd',
        'ids': ','.join(c['id'] for c in crypto_list),
        'price_change_percentage': '24h',
        'x_cg_demo_api_key': API_KEY
    }

    async with aiohttp.ClientSession() as session:
        async with session.get(API_URL, params=params, ssl=False)
as response:
        data = await response.json()
        return data

def save_prices(data: list[dict]):
    db: Session = SessionLocal()
    try:
        for coin in data:
            result = TrackingResult(
                coin_id=coin['id'],
                price=coin['current_price'],
                timestamp=datetime.utcnow()
            )
            db.add(result)
        db.commit()
        print("✅ Цены сохранены в БД")
    except Exception as e:
        print(f"❌ Ошибка сохранения в БД: {e}")

```

```

db.rollback()

finally:
    db.close()

async def sync_price():
    data = await fetch_crypto_data()
    if data:
        save_prices(data)
    else:
        print("⚠ Данные с CoinGecko не получены")

if __name__ == "__main__":
    asyncio.run(sync_price())

```

Приложение А.4

Файл «models.py»

```

import os
from datetime import datetime

import dotenv
from sqlalchemy import create_engine, Column, Integer, String,
Float, DateTime, ARRAY
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session

# Загружаем переменные окружения
dotenv.load_dotenv()

POSTGRES_USER = os.getenv("POSTGRES_USER")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD")
POSTGRES_NAME = os.getenv("POSTGRES_NAME")
POSTGRES_ADDRESS = os.getenv("POSTGRES_ADDRESS")
POSTGRES_PORT = os.getenv("POSTGRES_PORT", "5432")

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

DATABASE_URL = (
    f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}"
    f"@{POSTGRES_ADDRESS}:{POSTGRES_PORT}/{POSTGRES_NAME}"
)

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(bind=engine, autocommit=False,
autoflush=False)

Base = declarative_base()

class TrackingRuleDB(Base):
    __tablename__ = "tracking_rules"

    chat_id = Column(Integer, primary_key=True, index=True)
    period = Column(String(1), nullable=False) # 'm', 'h', 'd'
    coins = Column(ARRAY(String), nullable=False)
    last_notified_at = Column(DateTime, nullable=True)

class TrackingResult(Base):
    __tablename__ = "tracking_results"

    id = Column(Integer, primary_key=True, index=True)
    coin_id = Column(String, nullable=False)
    price = Column(Float, nullable=False)
    timestamp = Column(DateTime, default=datetime.utcnow)

# --- Функции для работы с БД ---

def save_prices(data: list[dict]):
    """
    Сохраняет список словарей с данными по монетам в
    tracking_results.
    Ожидается, что data = [{'id': 'bitcoin', 'current_price':
    12345.6}, ...]
    """

```

```

db: Session = SessionLocal()
try:
    for coin in data:
        result = TrackingResult(
            coin_id=coin['id'],
            price=coin['current_price'],
            timestamp=datetime.utcnow()
        )
        db.add(result)
    db.commit()
except Exception as e:
    print(f"Error saving prices to DB: {e}")
    db.rollback()
finally:
    db.close()

def get_tracking_rule(chat_id: int) -> TrackingRuleDB | None:
    """
    Получить правило отслеживания по chat_id.
    """
    db: Session = SessionLocal()
    try:
        rule =
db.query(TrackingRuleDB).filter(TrackingRuleDB.chat_id
chat_id).first()
        return rule
    finally:
        db.close()

def save_tracking_rule(chat_id: int, period: str, coins:
list[str]):
    """
    Создать или обновить правило отслеживания.
    """
    db: Session = SessionLocal()
    try:

```

```

rule = db.query(TrackingRuleDB).filter(TrackingRuleDB.chat_id ==
chat_id).first()
    if rule:
        rule.period = period
        rule.coins = coins
    else:
        rule = TrackingRuleDB(chat_id=chat_id, period=period,
coins=coins)
        db.add(rule)
    db.commit()
except Exception as e:
    print(f"Error saving tracking rule: {e}")
    db.rollback()
finally:
    db.close()

```

Приложение А.5

Файл «dto.py»

```

from pydantic import BaseModel, Field
from typing import List, Literal

class TrackingRule(BaseModel):
    period: Literal['m', 'h', 'd']
    chatId: int
    coins: List[str]

```

Приложение А.6

Файл «tg.py»

```

import dotenv
import os
import aiohttp

```

```

dotenv.load_dotenv()
bot_token = os.getenv('TG_BOT_TOKEN')

async def send_message(chat_id: str, text: str):
    url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
    payload = {
        "chat_id": chat_id,
        "text": text
    }

    async with aiohttp.ClientSession() as session:
        async with session.post(url, json=payload) as response:
            if response.status != 200:
                error = await response.text()
                print(f"❌ Ошибка отправки сообщения: {error}")
            else:
                print(f"✅ Сообщение отправлено в {chat_id}")

```

Приложение А.7

Файл «notifier.py»

```

import logging

from sqlalchemy.orm import Session
from datetime import datetime, timedelta
from app.models import SessionLocal, TrackingRuleDB, TrackingResult
from app.tg import send_message
import asyncio

PERIOD_TO_MINUTES = {

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

'm': 1,
'h': 60,
'd': 1440
}

async def notify_users():
    print('Notifying users')
    db: Session = SessionLocal()

    try:
        rules = db.query(TrackingRuleDB).all()
        now = datetime.utcnow()

        for rule in rules:
            minutes = PERIOD_TO_MINUTES.get(rule.period, 1)
            time_threshold = now - timedelta(minutes=minutes)

            # Только если пора уведомить снова
            if rule.last_notified_at is None or
rule.last_notified_at < time_threshold:
                for coin in rule.coins:
                    recent = (
                        db.query(TrackingResult)
                        .filter(TrackingResult.coin_id == coin)
                        .filter(TrackingResult.timestamp >=
time_threshold)

                    .order_by(TrackingResult.timestamp.desc())
                    .first()

                    )

                    if recent:
                        msg = (
                            f"🔔 Монета: {coin}\n"

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

```

f"💰 Текущая цена: ${recent.price}\n"
        )
        await send_message(str(rule.chat_id),
msg)

        # Обновляем время последнего уведомления
        rule.last_notified_at = now
        db.commit()

except Exception as e:
    print(f"[ERROR] notify_users: {e}")
finally:
    db.close()

```

Приложение А.8

Файл «requirements.txt»

```

aiohappyeyeballs==2.6.1
aiohttp==3.12.2
aiosignal==1.3.2
annotated-types==0.7.0
anyio==4.9.0
APScheduler==3.11.0
attrs==25.3.0
click==8.2.1
fastapi==0.115.12
frozenlist==1.6.0
h11==0.16.0
idna==3.10
multidict==6.4.4
propcache==0.3.1
psycpg2-binary==2.9.10
pydantic==2.11.5
pydantic_core==2.33.2
python-dotenv==1.1.0

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```
sniffio==1.3.1
SQLAlchemy==2.0.41
starlette==0.46.2
typing-inspection==0.4.1
typing_extensions==4.13.2
tzlocal==5.3.1
uvicorn==0.34.2
yarl==1.20.0
```

					<i>ПГУ 1.090301.18.001 ПЗ</i>	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б

Листинги frontend-части

Приложение Б.1

Файл «App.vue»

```
<template>
  <div class="app-container">
    <router-view v-slot="{ Component }">
      <transition name="fade" mode="out-in">
        <component :is="Component"/>
      </transition>
    </router-view>

    <Navbar/>
  </div>
</template>

<script>
import Navbar from './components/Navbar.vue'

export default {
  name: 'App',
  components: {Navbar},

  mounted() {
    if (window.Telegram && window.Telegram.WebApp) {
      const tg = window.Telegram.WebApp;
      const user = tg.initDataUnsafe?.user;

      if (user) {
        localStorage.setItem('telegram_user_id', user.id);
      } else {
        localStorage.setItem('telegram_user_id', '730072130');
      }
    }
  }
}
```

					ПГУ 1.090301.18.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    }

    }

}

</script>

<style>
.app-container {
  max-width: 100%;
  margin: 0 auto;
  min-height: 100vh;
  background: var(--bg-primary);
  font-family: -apple-system, BlinkMacSystemFont, sans-serif;
}

.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.2s ease;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}
</style>

```

Приложение Б.2

Файл «Navbar.vue»

```

<template>
  <nav class="navbar">
    <router-link
      v-for="route in routes"

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

:key="route.name"
      :to="route.path"
      class="nav-link"
      active-class="active"
    >
      <span class="nav-icon">{{ route.meta.icon }}</span>
      <span class="nav-text">{{ route.meta.title }}</span>
    </router-link>
  </nav>
</template>

<script>
import {computed} from 'vue'
import {useRouter} from 'vue-router'

export default {
  name: "Navbar",
  setup() {
    const router = useRouter()
    const routes = computed(() =>
      router.getRoutes().filter(r => r.meta.icon)
    )

    return {routes}
  }
}
</script>

<style scoped>
.navbar {
  display: flex;
  justify-content: space-around

```

					ПГУ 1.090301.18.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```

align-items: center;
background: var(--bg-primary);
backdrop-filter: blur(10px);
border-top: 1px solid rgba(0, 0, 0, 0.1);
position: fixed;
bottom: 0;
padding-bottom: 35px;
padding-top: 5px;
left: 0;
right: 0;
z-index: 100;
}

```

```

.nav-link {
display: flex;
flex-direction: column;
align-items: center;
text-decoration: none;
color: #666;
padding: 8px 12px;
border-radius: 8px;
transition: all 0.2s;
}

```

```

.nav-link.active {
color: #007AFF;
background: rgba(0, 122, 255, 0.1);
}

```

```

.nav-icon {
font-size: 20px;
margin-bottom: 2px;
}

```

```

.nav-text {

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

```
font-size: 12px;
  font-weight: 500;
}
</style>
```

Приложение Б.3

Файл «CryptoCard.vue»

```
<template>
  <div class="crypto-card" @click="handleClick">
    <div class="crypto-icon">
      
      <div class="icon-fallback" v-else>
        {{ crypto.symbol.toUpperCase().slice(0, 2) }}
      </div>
    </div>

    <div class="crypto-info">
      <span class="crypto-symbol">{{ crypto.symbol.toUpperCase()
    }}</span>
      <span class="crypto-name">{{ crypto.name }}</span>
    </div>

    <div class="crypto-price">
      <span class="price">
        $ {{ crypto.current_price.toLocaleString('ru-RU') }}
      </span>
      <span
        class="price-change"
        :class="{ positive: crypto.price_change_percentage_24h
        >= 0, negative: crypto.price_change_percentage_24h < 0 }"
```

```

>
      {{crypto.price_change_percentage_24h >= 0 ? '↑' : '↓'
      }}
Math.abs(crypto.price_change_percentage_24h).toFixed(2) }}%
    </span>
  </div>
</div>
</template>

<script setup>
import {ref} from 'vue'

const props = defineProps({
  crypto: {
    type: Object,
    required: true
  }
})

const emit = defineEmits(['click'])

const showImage = ref(true)
const imageError = ref(false)

const handleClick = () => {
  emit('click', props.crypto.id)
}

const handleImageError = () => {
  imageError.value = true
}
</script>

<style scoped>
.crypto-card {

```

```
display: flex;
  align-items: center;
  padding: 12px 16px;
  background: var(--bg-secondary);
  border-radius: 12px;
  margin-bottom: 8px;
  cursor: pointer;
  transition: transform 0.2s;
  gap: 12px;
}
```

```
.crypto-card:active {
  transform: scale(0.98);
}
```

```
.crypto-icon {
  width: 36px;
  height: 36px;
  border-radius: 50%;
  overflow: hidden;
  background: var(--bg-primary);
  display: flex;
  align-items: center;
  justify-content: center;
}
```

```
.crypto-icon img {
  width: 100%;
  height: 100%;
  object-fit: contain;
}
```

```
.icon-fallback {
  font-weight: bold;
}
```

					ИГУ 1.090301.18.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

```

font-size: 14px;
  color: var(--text-primary);
}

.crypto-info {
  flex: 1;
  display: flex;
  flex-direction: column;
  overflow: hidden;
}

.crypto-symbol {
  font-weight: 600;
  font-size: 16px;
  color: var(--text-primary);
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.crypto-name {
  font-size: 12px;
  color: var(--text-secondary);
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.crypto-price {
  display: flex;
  flex-direction: column;
  align-items: flex-end;
}

.price {
  font-weight: 600;

```

```

font-size: 16px;
  color: var(--text-primary);
}

.price-change {
  font-size: 12px;
  margin-top: 2px;
}

.positive {
  color: #34C759;
}

.negative {
  color: #FF3B30;
}

</style>

```

Приложение Б.4

Файл «HomeView.vue»

```

<template>
  <div class="home-view">
    <h1 class="title">Курс криптовалют</h1>
    <div v-if="loading" class="status-message">Загрузка...</div>
    <div v-else-if="error" class="status-message error">{{ error
    }}</div>
    <div v-else class="crypto-list">
      <CryptoCard
        v-for="crypto in cryptoList"
        :key="crypto.id"
        :crypto="crypto"
      />
    </div>
    <div class="last-updated">
      Обновлено: {{ lastUpdated }}
    </div>
  </div>

```

```

</div>
  </div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import axios from 'axios'
import CryptoCard from '@components/CryptoCard.vue'

const cryptoList = ref([
  { id: 'bitcoin', symbol: 'btc', name: 'Bitcoin',
current_price: 0, price_change_percentage_24h: 0 },
  { id: 'ethereum', symbol: 'eth', name: 'Ethereum',
current_price: 0, price_change_percentage_24h: 0 },
  { id: 'tether', symbol: 'usdt', name: 'Tether', current_price:
0, price_change_percentage_24h: 0 },
  { id: 'solana', symbol: 'sol', name: 'Solana', current_price:
0, price_change_percentage_24h: 0 },
  { id: 'tron', symbol: 'trx', name: 'Tron', current_price: 0,
price_change_percentage_24h: 0 }
])
const loading = ref(true)
const error = ref(null)
const lastUpdated = ref('')
let intervalId = null
const chatId = ref(localStorage.getItem('telegram_chat_id'))

const API_KEY = 'CG-SK4VWzfr73G4CmXjfXuujvL4'
const API_URL = 'https://api.coingecko.com/api/v3/coins/markets'

const fetchData = async () => {
  try {
    loading.value = true
    error.value = null

    const response = await axios.get(API_URL, {

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

params: {
  vs_currency: 'usd',
  ids: cryptoList.value.map(c => c.id).join(','),
  price_change_percentage: '24h',
  x_cg_demo_api_key: API_KEY
}
}))

cryptoList.value = cryptoList.value.map(coin => {
  const updatedData = response.data.find(c => c.id ===
coin.id) || {}
  return {
    ...coin,
    current_price: updatedData.current_price || 0,
    price_change_percentage_24h:
updatedData.price_change_percentage_24h || 0,
    image: updatedData.image,

  }
}))

const now = new Date()
lastUpdated.value = now.toLocaleTimeString()

} catch (err) {
  error.value = 'Failed to load data from API. Please try
again later.'
  console.error('API error:', err)
} finally {
  loading.value = false
}
}

const startPolling = () => {

```

```

fetchData()
  intervalId = setInterval(fetchData, 60000)
}

onMounted(startPolling)

onUnmounted(() => {
  if (intervalId) {
    clearInterval(intervalId)
  }
})
</script>

```

```

<style scoped>
.home-view {
  padding: 16px;
  min-height: calc(100vh - 99.5px);
  background: var(--bg-primary);
}

```

```

.title {
  color: var(--text-primary);
  text-align: center;
  margin-bottom: 20px;
}

```

```

.status-message {
  text-align: center;
  padding: 20px;
  color: var(--text-secondary);
}

```

```

.status-message.error {
  color: #ff3b30;
}

```

```

.crypto-list {

```

					ПГУ 1.090301.18.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

display: flex;
  flex-direction: column;
  gap: 12px;
}

.last-updated {
  text-align: center;
  margin-top: 20px;
  color: var(--text-secondary);
  font-size: 0.8rem;
}
</style>

```

Приложение Б.5

Файл «ConverterView.vue»

```

<template>
  <div class="converter-view">
    <h1 class="title">Конвертер валют</h1>
    <div class="converter-card">
      <div class="input-group">
        <input
          v-model.number="amount"
          type="number"
          placeholder="0.00"
          class="amount-input"
          min="0"
          max="1e15"
          @input="handleInput"
        />

        <div class="custom-select"
          @click="toggleDropdown('from')">
          <span>{{ getSymbol(fromCurrency) }}</span>
          <svg class="arrow" :class="{ open: dropdownOpen.from
            }" viewBox="0 0 24 24" width="16" height="16">
            <path d="M7 10 5 5 15 5" fill="currentColor"/>

```

```
</svg>
```

```
<ul v-if="dropdownOpen.from" class="dropdown-list">
  <li
    v-for="coin in currencyOptions"
    :key="coin.id"
    @click.stop="selectCurrency('from', coin.id)"
    :class="{ selected: fromCurrency === coin.id }"
  >
    {{ coin.symbol.toUpperCase() }}
  </li>
</ul>
</div>
</div>
```

```
<div class="swap-icon" @click="swapCurrencies"
title="Поменять местами">🔄</div>
```

```
<div class="input-group">
  <input
    :value="convertedAmount"
    type="text"
    placeholder="0.00"
    class="amount-input"
    readonly
  />
</div>
```

```
<div class="custom-select"
@click="toggleDropdown('to')">
  <span>{{ getSymbol(toCurrency) }}</span>
  <svg class="arrow" :class="{ open: dropdownOpen.to }"
viewBox="0 0 24 24" width="16" height="16">
    <path d="M7 10 5 5 5 5z" fill="currentColor"/>
  </svg>
  <ul v-if="dropdownOpen.to" class="dropdown-list">
```

```

<li
    v-for="coin in currencyOptions"
    :key="coin.id"
    @click.stop="selectCurrency('to', coin.id)"
    :class="{ selected: toCurrency === coin.id }"
  >
    {{ coin.symbol.toUpperCase() }}
  </li>
</ul>
</div>
</div>

<div class="conversion-info">
  <div v-if="conversionRate" class="rate">
    1  {{ getSymbol(fromCurrency).toUpperCase() }} = {{
conversionRate }} {{ getSymbol(toCurrency).toUpperCase() }}
  </div>
  <div v-if="error" class="error-message">{{ error
}}</div>
  <div v-if="loading" class="loading-
message">Загрузка...</div>
</div>
</div>
</template>

<script setup>
import {ref, onMounted, watch} from 'vue'
import axios from 'axios'

const API_URL = 'https://api.coingecko.com/api/v3/simple/price'

const amount = ref(1)
const fromCurrency = ref('bitcoin')
const toCurrency = ref('usd')
const convertedAmount = ref('0.00')

```

```

const conversionRate = ref(null)
const error = ref(null)
const loading = ref(false)
const dropdownOpen = ref({from: false, to: false})
const rates = ref({})

// Валюты для выбора
const currencyOptions = [
  {id: 'bitcoin', symbol: 'BTC'},
  {id: 'ethereum', symbol: 'ETH'},
  {id: 'usd', symbol: 'USD'},
  {id: 'solana', symbol: 'SOL'},
  {id: 'tron', symbol: 'TRX'},
]

const getSymbol = (id) => {
  const coin = currencyOptions.find(c => c.id === id)
  return coin ? coin.symbol : id
}

const handleInput = (e) => {
  const value = parseFloat(e.target.value)
  if (isNaN(value) || value < 0) {
    amount.value = 0
  } else if (value > 1e15) {
    amount.value = 1e15
    error.value = 'Введено максимально допустимое значение'
  } else {
    amount.value = value
    error.value = null
  }
  convertCurrency()
}

const formatNumber = (num) => {

```

```

if (num === 0) return '0.00'

// Для очень больших чисел
if (num >= 1e6) {
  return num.toExponential(4)
}

// Определяем количество знаков после запятой
if (num >= 1000) {
  return num.toLocaleString(undefined, {maximumFractionDigits:
2})
} else if (num >= 1) {
  return num.toLocaleString(undefined, {maximumFractionDigits:
4})
} else if (num >= 0.01) {
  return num.toLocaleString(undefined, {maximumFractionDigits:
6})
} else {
  return num.toLocaleString(undefined, {maximumFractionDigits:
8})
}
}

const fetchRates = async () => {
  try {
    loading.value = true
    error.value = null

    const ids = currencyOptions.map(c => c.id).join(',')
    const response = await axios.get(API_URL, {
      params: {
        ids,
        vs_currencies: 'usd'
      }
    })
  })
}

```

```

const pricesUSD = response.data
  rates.value = {}

  currencyOptions.forEach(c1 => {
    rates.value[c1.id] = {}
    currencyOptions.forEach(c2 => {
      if (c1.id === c2.id) {
        rates.value[c1.id][c2.id] = 1
      } else if (c1.id === 'usd') {
        rates.value[c1.id][c2.id] = 1 / (pricesUSD[c2.id]?.usd
|| 1)

      } else if (c2.id === 'usd') {
        rates.value[c1.id][c2.id] = pricesUSD[c1.id]?.usd || 1
      } else {
        rates.value[c1.id][c2.id] = (pricesUSD[c1.id]?.usd ||
1) / (pricesUSD[c2.id]?.usd || 1)
      }
    })
  })

} catch (err) {
  error.value = 'Ошибка загрузки данных. Попробуйте позже.'
  console.error('API error:', err)
} finally {
  loading.value = false
}
}

const convertCurrency = () => {
  try {
    error.value = null

    if (amount.value > Number.MAX_SAFE_INTEGER) {
      error.value = 'Слишком большое число'
      convertedAmount.value = '∞'
    }
  }
}

```

```

conversionRate.value = null
    return
}

if (!rates.value[fromCurrency.value] ||
!rates.value[fromCurrency.value][toCurrency.value]) {
    convertedAmount.value = '0.00'
    conversionRate.value = null
    return
}

if (fromCurrency.value === toCurrency.value) {
    convertedAmount.value = formatNumber(amount.value)
    conversionRate.value = '1.000000'
    return
}

Constrate= rates.value[fromCurrency.value][toCurrency.value]
conversionRate.value = formatNumber(rate)
const result = amount.value * rate

convertedAmount.value = formatNumber(result)

} catch (err) {
    error.value = 'Ошибка конвертации. Попробуйте позже.'
    console.error('Conversion error:', err)
}

}

const swapCurrencies = () => {
    const temp = fromCurrency.value
    fromCurrency.value = toCurrency.value
    toCurrency.value = temp
    convertCurrency()
}

```

```

const toggleDropdown = (which) => {
  dropdownOpen.value[which] = !dropdownOpen.value[which]
  if (which === 'from') dropdownOpen.value.to = false
  if (which === 'to') dropdownOpen.value.from = false
}

const selectCurrency = (which, id) => {
  if (which === 'from') fromCurrency.value = id
  else toCurrency.value = id

  dropdownOpen.value[which] = false
  convertCurrency()
}

const closeDropdowns = (e) => {
  if (!e.target.closest('.custom-select')) {
    dropdownOpen.value.from = false
    dropdownOpen.value.to = false
  }
}

onMounted(async () => {
  document.addEventListener('click', closeDropdowns)
  await fetchRates()
  convertCurrency()
})

watch([fromCurrency, toCurrency], convertCurrency)
</script>

<style scoped>
:root {
  --bg-primary: #f9f9f9;
  --bg-secondary: #fff;
  --border-color: #ddd;
  --text-primary: #111;

```

```

--text-secondary: #555;
  --accent-color: #007aff;
}

.converter-view {
  padding: 16px;
  background: var(--bg-primary);
  min-height: 100vh;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI",
Roboto, Oxygen,
  Ubuntu, Cantarell, "Open Sans", "Helvetica Neue", sans-serif;
}

.title {
  color: var(--text-primary);
  text-align: center;
  margin-bottom: 20px;
  font-weight: 600;
  font-size: 28px;
}

.converter-card {
  max-width: 500px;
  margin: 0 auto;
  background: var(--bg-secondary);
  border-radius: 20px;
  padding: 24px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}

.input-group {
  display: flex;
  margin-bottom: 20px;
  border: 1px solid var(--border-color);
  border-radius: 16px;

```

```

overflow: hidden;
  align-items: center;
}

.amount-input {
  flex: 1;
  padding: 14px 16px;
  border: none;
  background: var(--bg-secondary);
  color: var(--text-primary);
  font-size: 18px;
  font-weight: 600;
  outline-offset: 2px;
  outline-color: transparent;
  transition: outline-color 0.2s ease;
}

.amount-input:focus {
  outline-color: var(--accent-color);
}

.custom-select {
  padding: 14px 16px;
  position: absolute;
  right: 10dvw;
  border-top-right-radius: 16px;
  border-bottom-right-radius: 16px;
  background: var(--bg-secondary);
  border: 1px solid var(--border-color);
  cursor: pointer;
  user-select: none;
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-weight: 700;
  font-size: 18px;

```

```

color: var(--text-primary);
  transition: border-color 0.3s ease, box-shadow 0.3s ease;
  min-width: 90px;
}

.arrow {
  transition: transform 0.3s ease;
  fill: var(--accent-color);
  margin-left: 8px;
}

.arrow.open {
  transform: rotate(180deg);
}

.dropdown-list {
  position: absolute;
  top: 100%;
  right: 0;
  width: 120px;
  background: var(--bg-secondary);
  border-radius: 16px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.12);
  max-height: 160px;
  overflow-y: auto;
  z-index: 10;
  padding: 0;
  margin: 4px 0 0 0;
  list-style: none;
  font-weight: 600;
  font-size: 16px;
}

.dropdown-list li {
  padding: 12px 16px;
  cursor: pointer;

```

```

transition: background-color 0.2s ease;
  color: var(--text-primary);
}

.dropdown-list li:hover {
  background: var(--accent-color);
  color: white;
}

.dropdown-list li.selected {
  background: rgba(0, 122, 255, 0.1);
  color: var(--accent-color);
}

.swap-icon {
  text-align: center;
  margin: 8px 0 20px 0;
  cursor: pointer;
  font-size: 28px;
  user-select: none;
  transition: transform 0.3s ease, color 0.3s ease;
  color: var(--accent-color);
}

.swap-icon:hover {
  color: #0051a8;
  transform: rotate(180deg);
}

.conversion-info {
  margin-top: 16px;
  text-align: center;
  font-weight: 600;
  font-size: 16px;
  color: var(--text-secondary);
}

```

```

.rate {
  margin-bottom: 8px;
}

.error-message {
  color: #ff3b30;
  margin-top: 8px;
}

.loading-message {
  color: var(--accent-color);
  margin-top: 8px;
}
</style>

```

Приложение Б.6

Файл «NotificationsView.vue»

```

<template>
  <div class="notification-settings">
    <h1 class="title">Уведомления</h1>

    <div class="section">
      <h2 class="section-title">Отслеживаемые монеты</h2>
      <div class="coin-list">
        <div
          v-for="coin in availableCoins"
          :key="coin.id"
          class="coin-item"
          @click="toggleCoinNotification(coin.id)"
        >
          <span class="coin-name">{{ coin.name }} {{
            coin.symbol.toUpperCase() }}</span>
          <span class="toggle" :class="{ active:
            isCoinTracked(coin.id) }">
            <span class="toggle-handle"></span>
          </span>
        </div>
      </div>
    </div>
  </div>

```

```

</div>
    </div>
</div>

<div class="section">
    <h2 class="section-title">Настройки уведомлений</h2>
    <div class="frequency-options">
        <div
            v-for="option in frequencyOptions"
            :key="option.value"
            class="frequency-option"
            :class="{ active: frequency === option.value }"
            @click="frequency = option.value"
        >
            <span class="option-label">{{ option.label }}</span>
        </div>
    </div>
</div>

<button class="save-button" @click="saveSettings">
    Сохранить настройки
</button>

<div v-if="message" class="message" :class="{ error: isError
}">
    {{ message }}
</div>
</div>
</template>

<script setup>
import {onMounted, ref} from 'vue'

const domain = import.meta.env.VITE_API_BASE_URL

const availableCoins = ref([
    {id: 'bitcoin', symbol: 'btc', name: 'Bitcoin'},

```

```

{id: 'ethereum', symbol: 'eth', name: 'Ethereum'},
  {id: 'tether', symbol: 'usdt', name: 'Tether'},
  {id: 'solana', symbol: 'sol', name: 'Solana'},
  {id: 'tron', symbol: 'trx', name: 'Tron'},
])

const frequencyOptions = ref([
  {value: 'm', label: 'Каждую минуту'},
  {value: 'h', label: 'Каждый час'},
  {value: 'd', label: 'Раз в день'}
])

const trackedCoins = ref([])
const frequency = ref('h')
const message = ref('')
const isError = ref(false)
const chatId = ref('')

const parseParams = (params) => {
  if (params.period && frequencyOptions.value.find(o => o.value
=== params.period)) {
    frequency.value = params.period
  }
  if (Array.isArray(params.coins)) {
    const validCoins = params.coins.filter(c =>
      availableCoins.value.some(ac => ac.id === c)
    )
    trackedCoins.value = validCoins
  }
  if (params.chatId) {
    chatId.value = String(params.chatId)
  }
}

const loadSettings = async () => {

```

```

if (!chatId.value) return

try {
    const response = await
fetch(`${domain}/track/${chatId.value}`)

    if (response.status === 404) {
        console.warn('Настройки не найдены (404), используем
параметры из GET или дефолты')
        return
    }

    if (!response.ok) {
        throw new Error(`Ошибка загрузки настроек:
${response.status}`)
    }

    const data = await response.json()

    if (data.period && frequencyOptions.value.find(o => o.value
=== data.period)) {
        frequency.value = data.period
    }

    if (Array.isArray(data.coins)) {
        trackedCoins.value = data.coins.filter(c =>
            availableCoins.value.some(ac => ac.id === c)
        )
    }
} catch (error) {
    console.error('Ошибка при загрузке настроек:', error)
    isError.value = true
    setTimeout(() => {
        message.value = ''
    }, 1000)
}

```

```

    isError.value = false
      }, 3000)
    }
  }

const isCoinTracked = (coinId) => {
  return trackedCoins.value.includes(coinId)
}

const toggleCoinNotification = (coinId) => {
  if (isCoinTracked(coinId)) {
    trackedCoins.value = trackedCoins.value.filter(id => id !==
coinId)
  } else {
    trackedCoins.value.push(coinId)
  }
}

const saveSettings = async () => {
  if (!chatId.value) {
    message.value = 'Не удалось определить ID чата'
    isError.value = true
    setTimeout(() => {
      message.value = '';
      isError.value = false
    }, 3000)
    return
  }

  try {
    const response = await fetch(`${domain}/track`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },

```

```

body: JSON.stringify({
    period: frequency.value,
    chatId: chatId.value,
    coins: trackedCoins.value
})
}))

message.value = 'Настройки успешно сохранены!'
isError.value = false
} catch (error) {
    console.error('Ошибка сохранения:', error)
    message.value = 'Произошла ошибка: ' + error.message
    isError.value = true
} finally {
    setTimeout(() => {
        message.value = '';
        isError.value = false
    }, 3000)
}
}

onMounted(async () => {
    try {

        chatId.value = localStorage.getItem('telegram_user_id') ||
        ''

        const incomingParams = {}

        parseParams(incomingParams)

        if (chatId.value) {
            await loadSettings()
        }
    } catch (error) {
        console.error('Ошибка инициализации:', error)
    }
}

```

```

message.value = 'Ошибка инициализации'
  isError.value = true
  setTimeout(() => {
    message.value = '';
    isError.value = false
  }, 3000)
}
})
</script>

```

```

<style scoped>
/* Стили остаются такими же, как в предыдущем примере */
.notification-settings {
  background: var(--bg-primary);
  padding: 20px;
  height: calc(100vh - 99.5px);
  overflow: auto;
  max-width: 500px;
  margin: 0 auto;
}

.title {
  color: var(--text-primary);
  text-align: center;
  margin-bottom: 24px;
  margin-top: 0;
  font-size: 24px;
}

.section {
  margin-bottom: 24px;
}

.section-title {
  color: var(--text-primary);

```

Изм.	Лист	№ докум.	Подпись	Дата

```

font-size: 18px;
margin-bottom: 16px;
font-weight: 500;
}

.coin-list {
border: 1px solid var(--border-color);
border-radius: 12px;
overflow: hidden;
}

.coin-item {
display: flex;
justify-content: space-between;
align-items: center;
padding: 14px 16px;
border-bottom: 1px solid var(--border-color);
cursor: pointer;
transition: background-color 0.2s;
}

.coin-item:hover {
background-color: var(--bg-hover);
}

.coin-item:last-child {
border-bottom: none;
}

.coin-name {
color: var(--text-primary);
font-size: 15px;
}

.toggle {
position: relative;

```

```

width: 44px;
  height: 24px;
  border-radius: 12px;
  background-color: var(--bg-secondary);;
  transition: background-color 0.2s;
}

.toggle.active {
  background-color: #007AFF;
}

.toggle-handle {
  position: absolute;
  top: 2px;
  left: 2px;
  width: 20px;
  height: 20px;
  border-radius: 50%;
  background-color: white;
  transition: transform 0.2s;
}

.toggle.active .toggle-handle {
  transform: translateX(20px);
}

.frequency-options {
  display: grid;
  grid-template-columns: repeat(1, 1fr);
  gap: 12px;
}

.frequency-option {
  display: flex;
  flex-direction: column;
  align-items: center;

```

```
padding: 16px 8px;
border-radius: 12px;
background: var(--bg-primary);
border: 1px solid var(--border-color);
cursor: pointer;
transition: all 0.2s;
}

.frequency-option:hover {
background: var(--bg-hover);
}

.frequency-option.active {
border-color: #007AFF;
background: rgba(0, 122, 255, 0.1);
}

.option-label {
font-size: 13px;
text-align: center;
color: var(--text-primary);
}

.save-button {
width: 100%;
padding: 14px;
background: #007AFF;
color: white;
border: none;
border-radius: 12px;
margin-top: 24px;
cursor: pointer;
font-size: 16px;
font-weight: 500;
transition: opacity 0.2s;
}
```

```

.save-button:hover {
  opacity: 0.9;
}

.message {
  text-align: center;
  margin-top: 16px;
  font-size: 14px;
}

.message.error {
  color: #FF3B30;
}

.message:not(.error) {
  color: #34C759;
}
</style>

```

Приложение Б.7

Файл «index.js»

```

import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import ConverterView from '../views/ConverterView.vue'
import NotificationsView from '../views/NotificationsView.vue'

const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView,
    meta: { title: 'Курсы валют', icon: '🌐' },
  },
  {
    path: '/converter',

```

```

name: 'converter',
  component: ConverterView,
  meta: { title: 'Конвертер', icon: '🔄' },
},
{
  path: '/notifications',
  name: 'notifications',
  component: NotificationsView,
  meta: { title: 'Уведомления', icon: '🔔' }
}
]

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes
})

router.beforeEach((to) => {
  document.title = to.meta.title + ' | Crypto Tracker'
})

export default router

```

Приложение Б.8

Файл «main.js»

```

import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import axios from 'axios'
import './assets/main.css'

const app = createApp(App)

// Инициализация Telegram WebApp
if (window.Telegram?.WebApp) {

```

```

try {
    window.Telegram.WebApp.ready()
    window.Telegram.WebApp.expand()

    // Для тестирования в браузере
    if (!window.Telegram.WebApp.initDataUnsafe?.user) {
        window.Telegram.WebApp.initDataUnsafe = {
            user: { id: 123456789 } // Тестовый ID для
разработки
        }
    }
} catch (e) {
    console.error('Telegram WebApp init error:', e)
}

app.use(router)
app.mount('#app')

```

Приложение Б.9

Файл «vite.config.js»

```

import {fileURLToPath, URL} from 'node:url'

import {defineConfig} from 'vite'
import vue from '@vitejs/plugin-vue'
import vueDevTools from 'vite-plugin-vue-devtools'

// https://vite.dev/config/
export default defineConfig({
  plugins: [vue(), vueDevTools()],
  server: {
    host: '0.0.0.0',
    port: 5173,
  },
  preview: {
    allowedHosts: '*',

```

```
host: true,
    port: 5173,
  },
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src',
import.meta.url))
    },
  },
})
```

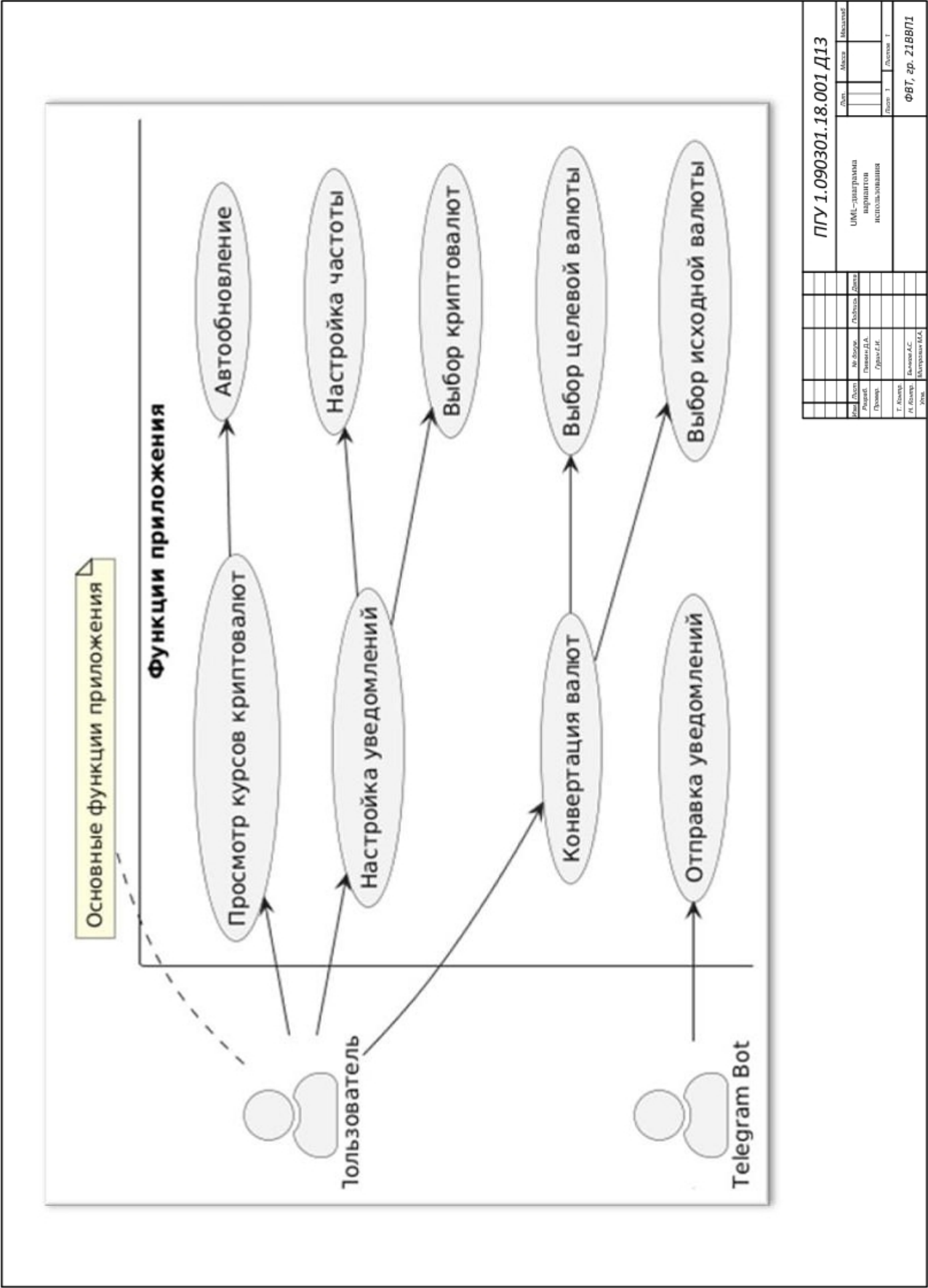
Приложение В

(обязательное)

Мини-приложение в Telegram для мониторинга и анализа криптовалютных
активов

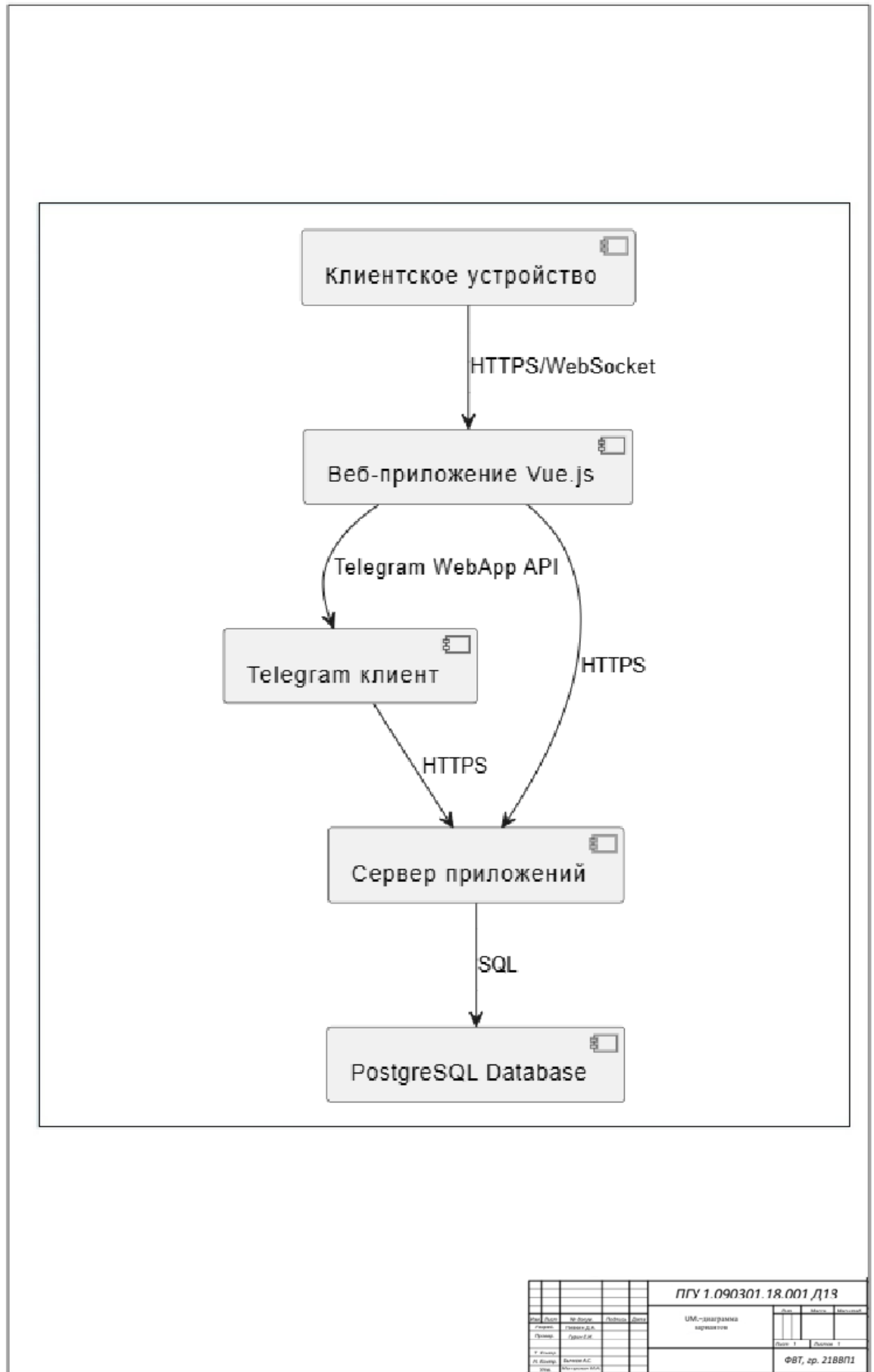
UML-диаграммы

					ПГУ 1.090301.18.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		



ПГУ 1.090301.18.001 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Рязань	Д.А.	Рязань	Д.А.	Рязань	Д.А.	Рязань	Д.А.	Рязань	Д.А.
Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань
Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань
Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань	Г. Рязань
ФБТ, стр. 2188/1									

Приложение В.2 – UML-диаграмма развертывания



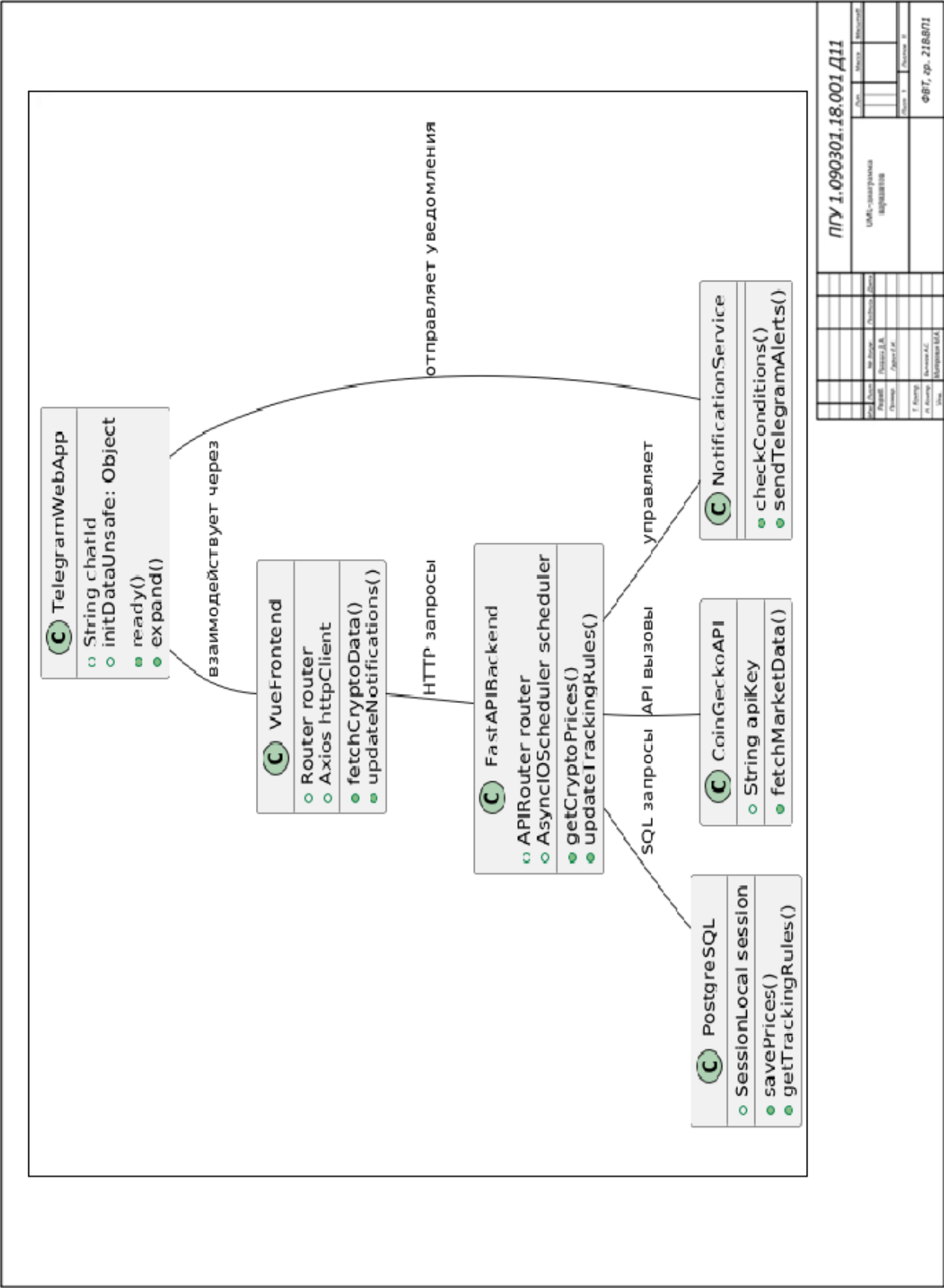
					ПГУ 1.090301.18.001 Л13		
Изм.	Лист	№ докум.	Подпись	Дата	UML-диаграмма приложения	Дата	Масштаб
						Листов 1	Листов 1
					ФБТ, зр. 2188П1		

Изм.	Лист	№ докум.	Подпись	Дата

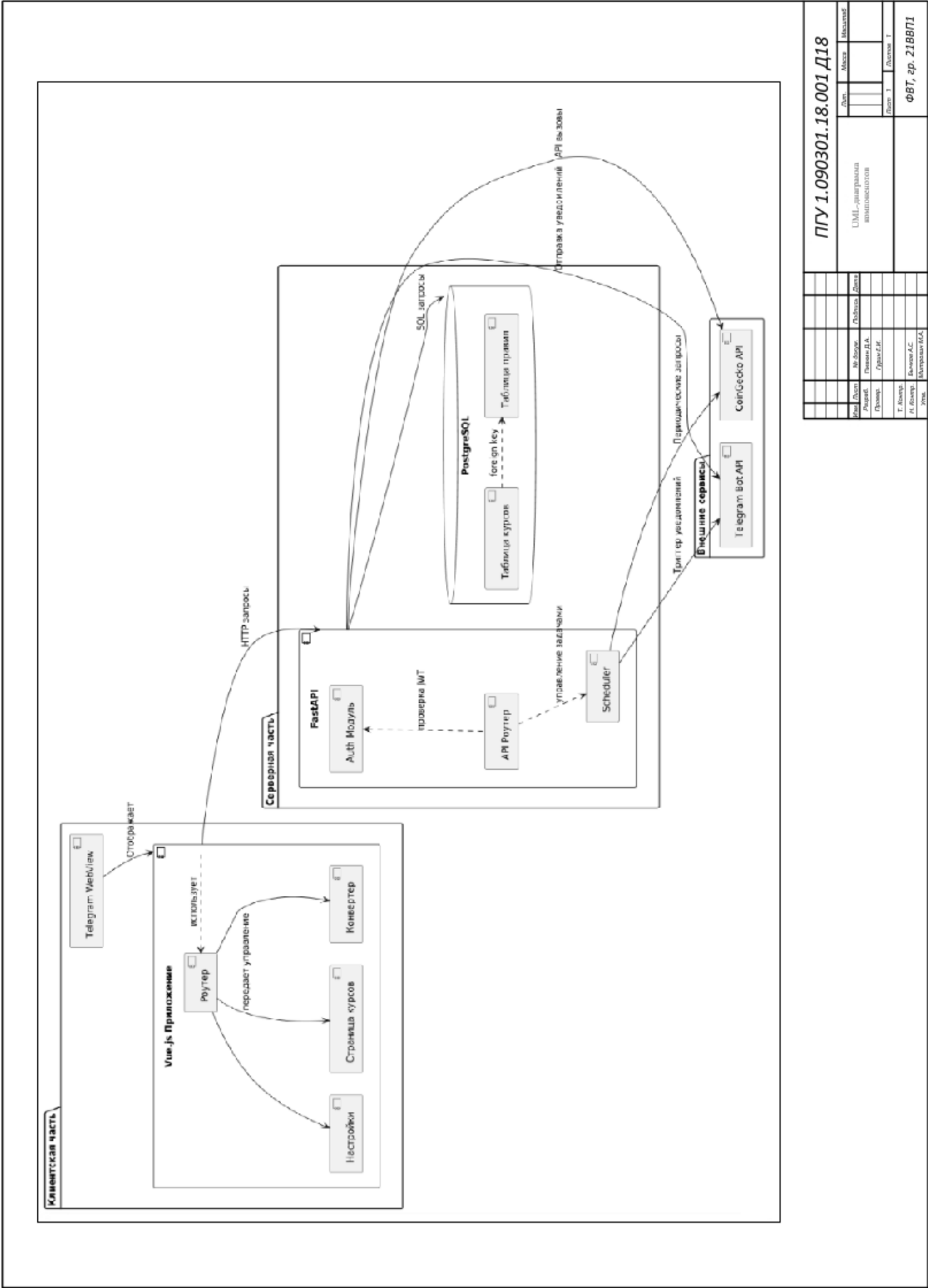
ПГУ 1.090301.18.001 ПЗ

Лист

116



ПГУ 1.090301.18.001 Д11				Дата	Имя	Подпись
UML-диаграмма приложения				Стр. 1	Стр. 2	Стр. 3
ФБТ, ФР. 2188/П						



ПУ 1.090301.18.001 Д18

Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Иван	Иванов	Петр	Петров	Сидор	Сидоров
Сидор	Сидоров	Петр	Петров	Иван	Иванов
Иван	Иванов	Петр	Петров	Сидор	Сидоров

Приложение Г

(обязательное)

Мини-приложение в Telegram для мониторинга и анализа криптовалютных

активов

Презентация

					ПГУ 1.090301.18.001 ПЗ	Лист
						119
Изм.	Лист	№ докум.	Подпись	Дата		

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

**МИНИ-ПРИЛОЖЕНИЕ В TELEGRAM ДЛЯ
МОНИТОРИНГА И АНАЛИЗА КРИПТОВАЛЮТНЫХ
АКТИВОВ**

АВТОР РАБОТЫ: СТУДЕНТ ГР 21ВВП1 ПИВКИН Д.А.
НАУЧНЫЙ РУКОВОДИТЕЛЬ: Д.Т.Н, ПРОФЕССОР ГУРИН Е.И.

Мини-приложение в Telegram для
мониторинга и анализа
криптовалютных активов

ЦЕЛИ И ЗАДАЧИ

УПРОЩЕНИЕ МОНИТОРИНГА КРИПТОРЫНКА

- Отображение текущих курсов с частым автоматическим обновлением
- Одновременное отслеживание Bitcoin, Ethereum и других топовых активов
- Расположение всех отслеживаемых криптовалют на одной странице

АВТОМАТИЗАЦИЯ УВЕДОМЛЕНИЙ

- Выбор желаемых криптовалют и периода времени получения оповещений о изменении цен
- Доставка уведомлений в Telegram без задержек
- История всех уведомлений в чате с ботом

ИНТЕГРАЦИЯ С TELEGRAM

- Вход через Telegram WebApp в один клик
- Одинаковый интерфейс на мобильных и десктопных версиях Telegram

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И АКТУАЛЬНОСТЬ

Проблемы традиционных методов мониторинга крипторынка

- ✓ Ручной сбор данных – пользователи вынуждены постоянно проверять курсы на разных сайтах, что отнимает время.
- ✓ Отсутствие персонализированных оповещений – без автоматических уведомлений легко пропустить важные изменения цен.

Потребности пользователей

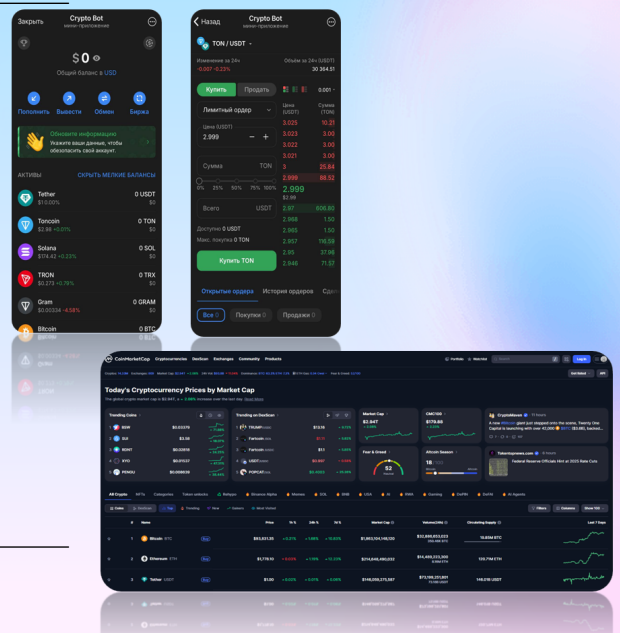
- ✓ Удобный доступ к данным – актуальные курсы и конвертация в одном месте.
- ✓ Интеграция с мессенджерами – получение информации там, где пользователи проводят больше времени (Telegram).

3

ОБЗОР АНАЛОГОВ: CRYPTOVOT И COINMARKETCAP

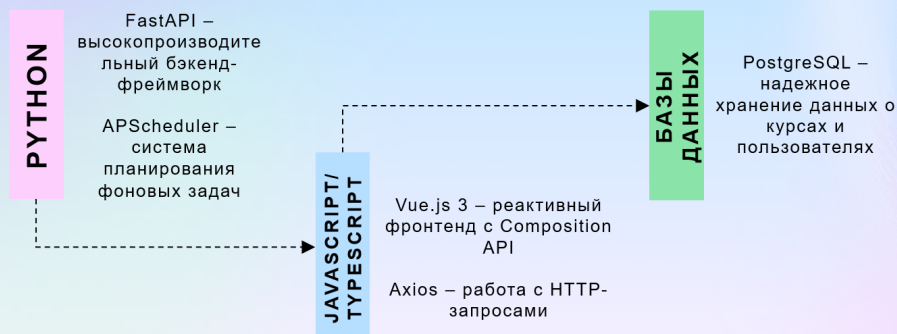
CRYPTOVOT – ПОЗВОЛЯЕТ ТОРГОВАТЬ, ОТСЛЕЖИВАТЬ КУРСЫ, И ПОЛУЧАТЬ УВЕДОМЛЕНИЯ, НО СЛОЖНЫЙ ИНТЕРФЕЙС ДЛЯ НОВИЧКОВ И ПЛАТНЫЕ ПОДПИСКИ

COINMARKETCAP – САМАЯ ПОЛНАЯ БАЗА ДАННЫХ О ТЫСЯЧАХ АКТИВОВ, РАСШИРЕННАЯ АНАЛИТИКА, НО ПРИСУТСТВУЕТ РЕКЛАМА И НЕТ ВОЗМОЖНОСТИ КОНВЕРТАЦИИ



4

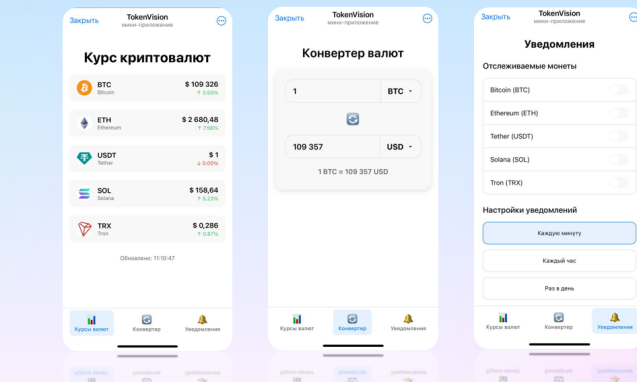
ТЕХНИЧЕСКИЙ ВЫБОР: ЯЗЫК И ФРЕЙМВОРКИ



5

ФУНКЦИОНАЛ МИНИ-ПРИЛОЖЕНИЯ

- ПРОСМОТР КУРСОВ ВАЛЮТ
- КОНВЕРТАЦИЯ
- УВЕДОМЛЕНИЯ



6

Изм.	Лист	№ докум.	Подпись	Дата

ЗАКЛЮЧЕНИЕ И ПЕРСПЕКТИВЫ РАЗВИТИЯ

Реализация

Мини-приложение TokenVision разработано как удобный инструмент для мониторинга крипторынка с интеграцией в Telegram, предоставляющий актуальные данные о курсах, автоматические уведомления и конвертацию валют.

Дальнейшее развитие

Возможность расширения аналитических функций, подключение новых API криптобирж, внедрение элементов торговли и выпуск мобильной версии, что превратит мини-приложение в полноценный финансовый инструмент для трейдеров.