Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»
Институт вычислительной математики и информационных технологий
Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 — Фундаментальная информатика и информационные технологии

Профиль: Системный анализ и информационные технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ИНТЕРАКТИВНАЯ СИСТЕМА ДЛЯ АВТОМАТИЗИРОВАННОГО ТРЕЙДИНГА И УПРАВЛЕНИЯ ФИНАНСОВЫМИ АКТИВАМИ

Обучающийся 4 курса
группы 09-132 (Чирков Л.С.)

Руководитель
канд. физ.-мат. наук, доцент (Андрианова А.А.)

Заведующий кафедрой системного анализа
и информационных технологий,
канд. физ-мат. наук, доцент (Васильев А.В.)

Казань – 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные финансовые рынки характеризуются высокой степенью волатильности, большим объемом информации и необходимостью принятия решений в условиях ограниченного времени. В таких условиях трейдерам всё сложнее оперативно обрабатывать и анализировать поступающие данные, а также своевременно принимать торговые решения. Оперативная торговля и отслеживание всевозможных изменений на рынке требует полной вовлеченности и постоянного присутствия около персонального компьютера или ноутбука, что не всегда является возможным.

С увеличением доступности API от брокерских платформ и расширением возможностей современных программных средств стало возможным объединение аналитических, информационных и торговых модулей в единую систему. Такие решения позволяют интегрировать сигнальные алгоритмы, технические индикаторы, базы данных и интерфейсы взаимодействия с пользователем в компактную и удобную форму. Пользователь получает не просто инструмент анализа, а полноценного помощника, способного в режиме реального времени отслеживать динамику финансовых инструментов, анализировать сигналы и предоставлять рекомендации или исполнять торговые операции по заданной логике.

Несмотря на то, что рынок содержит отдельные готовые решения, большинство из них ориентированы на профессиональных участников и требуют либо глубокой технической подготовки, либо серьёзных затрат времени на адаптацию под собственные задачи. Поэтому особый интерес представляет система, сочетающая простоту взаимодействия с широким набором функций, при этом оставаясь понятной для пользователя без опыта программирования или работы с крупными и тяжелыми терминалами.

В связи с этим возникает необходимость в создании интеллектуальной системы-помощника, способной автоматизировать процессы мониторинга, анализа и исполнения торговых операций.

Целью выпускной квалификационной работы является разработка программного комплекса, включающего интерфейс для взаимодействия пользователя с системой, механизмы для мониторинга и анализа рыночных данных, а также функциональность управления и настройки торговых сигналов и автоматических стратегий.

Для достижения поставленной цели были сформулированы следующие задачи:

1) определить общую архитектуру системы,
2) обозначить используемые технологии и определить пользовательские требования,
3) произвести обзор и сравнение с системами-аналогами,
4) реализовать функциональные компоненты системы,
5) реализовать пользовательский интерфейс системы,
6) интегрировать систему и API брокера для получения данных в реальном времени,
7) провести тестирование и эксплуатацию программных модулей,
8) провести анализ разработанной системы.

1. Общая схема системы

При проектировании системы необходимо учитывать общую структуру взаимодействия ее компонентов, поскольку это напрямую влияет на выбор используемых технологий и инструментов. На рисунке 1 представлена схема, которая отражает ключевые этапы обработки запросов пользователя и взаимодействия внутренних модулей системы.

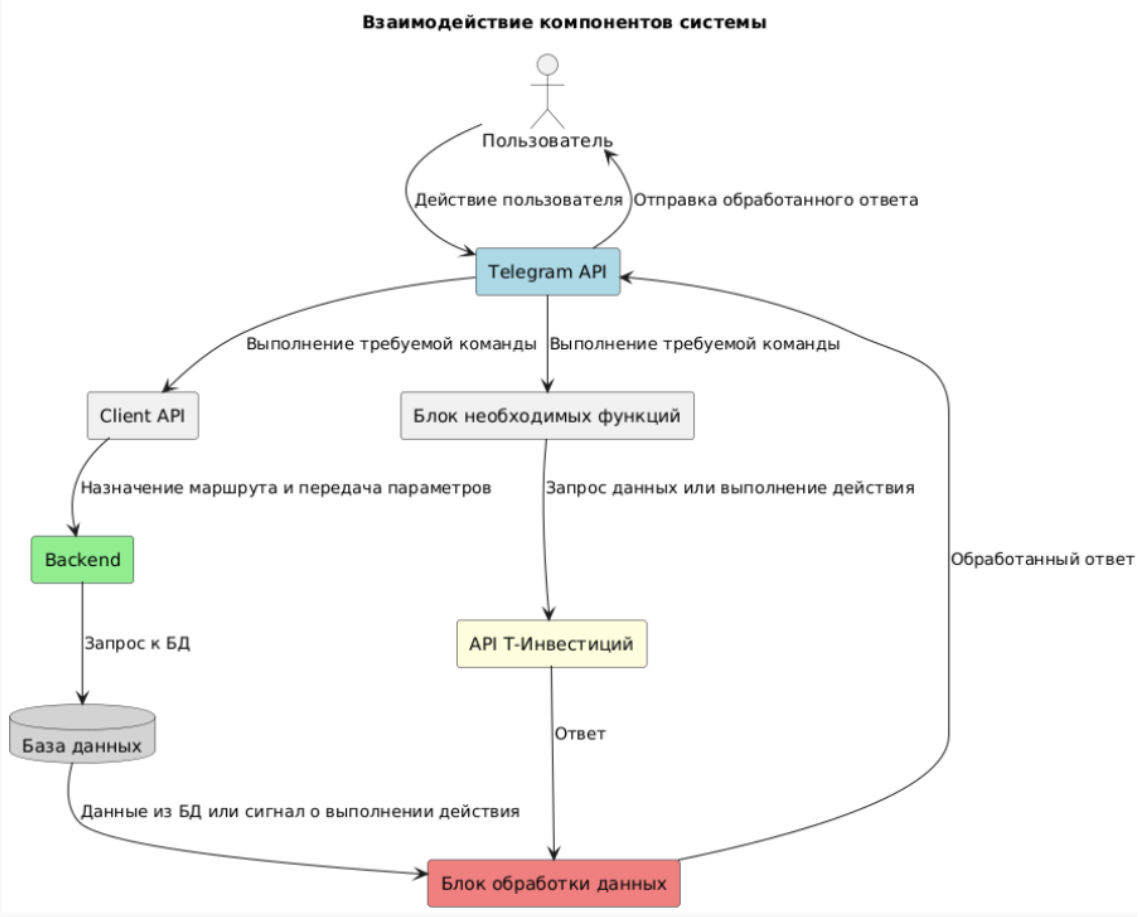

Рисунок 1 - Схема взаимодействия компонентов системы

Как можно увидеть из схемы, процесс работы системы включает несколько важных этапов:

– инициирование команды пользователем. Пользователь взаимодействует с Telegram-ботом, отправляя запрос или команду. Эти данные передаются через Telegram API с использованием библиотеки telebot. Запросы могут направляться либо к backend-части приложения через клиентскую прослойку, либо напрямую к методам API системы Т-Инвестиций через заранее разработанные функции;

– обработка backend-частью. Если запрос передается на backend, то выполняется обращение к базе данных и производится выполнение требуемой операции с возможным возвратом результата либо подтверждением завершения задачи;

– обращение к Т-Инвестициям. Если запрос направлен к API Т-Инвестиций, результатом может быть получение требуемых данных или выполнение операции. В пример можно привести возврат ценовых свечей или размещение торгового поручения;

– подготовка ответа пользователю. Независимо от маршрута обработки данных собранные данные приводятся в удобочитаемый вид и отправляются пользователю через Telegram-бот.

Данная схема детализирует взаимодействие между компонентами системы и показывает зависимости, которые необходимо учитывать при выборе технологий. Реализация такого взаимодействия требует использования инструментов, обеспечивающих стабильную работу различного рода подключений, а также соответствующих библиотек для обработки запросов к Telegram и внешним сервисам.

2. Используемые технологии

Выбор каждой технологии обоснован с точки зрения функциональности, удобства использования, масштабируемости и производительности. Основные инструменты включают язык программирования Python, систему управления базами данных SQLite, контейнеризацию с Docker, использование web-фреймворка FastAPI, взаимодействие с Telegram API с помощью библиотеки telebot, а также работу с API брокера Т-Инвестиции.

Python был выбран в качестве основного языка программирования, поскольку он может предоставить именно те наборы инструментов и библиотеки, которые необходимы для разработки такого рода приложений.

В качестве базы данных была выбрана SQLite, поскольку она встроена в Python и не требует отдельной установки и настройки [1]. Это позволяет пользователю быстро развернуть и запустить приложение без дополнительных манипуляций. Несмотря на свою легковесность, SQLite поддерживает все основные функции реляционной базы данных. Поскольку при разработке системы будет использоваться фреймворк FastAPI, если пользователь по какой-то причине захочет перейти на другое решение для управления базой данных, такое как PostgeSQL или MySQL, то это не составит для него труда, поскольку данный фреймворк поддерживает работу с различными базами данных через ORM-библиотеки, такие как SQLAlchemy и Tortoise-ORM [2]. Это позволяет легко изменить тип используемой базы данных, просто обновив параметры подключения и, при необходимости, адаптировав модели данных.

Для обеспечения удобного развертывания приложения на различных платформах используется контейнеризация с Docker. Этот инструмент позволяет создать изолированную среду, в которой приложение работает независимо от операционной системы и её конфигурации. Благодаря контейнерам можно избежать проблем, связанных с несовместимостью библиотек и зависимостей, а также упростить процесс развертывания и обновления системы. Docker также обеспечивает отказоустойчивость

приложения, предоставляя настраиваемые параметры автоматического перезапуска приложения в случае возникновения каких-либо ошибок [3].

Backend-система разработана на основе web-фреймворка FastAPI, который был выбран из-за своей высокой производительности, отсутствия лишних деталей и удобства разработки. Преимуществами данного фреймворка являются встроенная поддержка OpenAPI и автоматическая генерация документации, что упрощает тестирование и интеграцию с другими сервисами. Также FastAPI предоставляет удобные инструменты для валидации данных, снижая вероятность ошибок при их передаче [4].

Для взаимодействия с пользователем был выбран Telegram и библиотека telebot, поскольку Telegram является удобной платформой для автоматизации взаимодействия с пользователями, а библиотека telebot упрощает работу с API, предоставляя готовые методы для отправки сообщений и обработки различных команд.

API брокера Т-Инвестиции используется для получения рыночных данных, данных пользовательского портфеля и исполнения торговых операций. Данный брокер был выбран потому, что предоставляет открытое API с подробной документацией и готовые SDK для различных языков программирования, в частности для Python [5].

3. Пользовательские требования

Для построения эффективной системы первостепенной задачей является определение пользовательских требований. Эти требования определяют ожидаемый функционал, который система должна предоставлять в процессе работы. Основная цель — сделать использование системы максимально удобным, интуитивным и полезным для пользователя. Далее будут перечислены основные требования к функционалу системы:

– система должна предоставлять пользователю возможность получать состояние его инвестиционного портфеля, включая данные по каждому инструменту;

– необходимо, чтобы пользователь мог добавлять инструменты в персональную базу данных, получать список добавленных инструментов и удалять их при необходимости;

– система должна поддерживать функционал уведомлений, позволяя пользователю подписываться на обновления состояния рынка или изменения цен и отписываться от этих уведомлений при необходимости;

– пользователь должен иметь возможность получать информацию о состоянии рынка, включая обвалы или рост цен активов за определенный промежуток времени;

– система должна предоставить инструмент для настройки сигналов, которые отправляются пользователю на основании коэффициентов и параметров, выбранных вручную;

– система должна включать функционал торгового робота, который позволяет автоматизировать торговлю на основе настроенной стратегии. Пользователь должен иметь возможность гибко настраивать стратегию, выбирать временные промежутки для срабатывания сигналов, а также работать с реальным или тестовым счетом;

– система должна предоставлять отчеты по дивидендам за указанный период;

– пользователь должен иметь возможность получать построенные графики, которые будут отражать изменение цены и параметры сигналов;

– система должна включать базу знаний с краткими справками и инструкциями по настройке отдельных возможностей системы;

– пользователь должен иметь доступ к статистике операций, включая количество сделок, положительную и отрицательную маржу, а также аналитику совершения операций по сигналам, тикерам и времени.

Эти пользовательские требования составляют основу проектируемой системы и определяют набор функциональных компонентов, которые будут реализованы.

4. Обзор и сравнение с системами-аналогами

На текущий момент существуют решения в области автоматизации торговли на финансовых рынках, в том числе и с открытым исходным кодом. Однако при более глубоком изучении можно заметить, что такие решения имеют ряд недостатков, которые ограничивают их практическое применение. Главной проблемой подобных систем является отсутствие удобного и интуитивно понятного пользовательского интерфейса. В большинстве случаев взаимодействие с программой осуществляется напрямую через исходный код, что требует от пользователя определённых умений программирования. Например, настройку конфигурации для торгового робота в аналогичной программе можно увидеть на рисунке 2.

```
function getStrategyConfig(figi: string): StrategyConfig {
  return {
    /** ID инструмента */
    figi,
    /** По сколько лотов покупаем/продаем */
    orderLots: 1,
    /** Комиссия брокера, % от суммы сделки */
    brokerFee: 0.3,
    /** Интервал свечей */
    interval: CandleInterval.CANDLE_INTERVAL_5_MIN,
    /** Конфиг сигнала по отклонению текущей цены */
    profit: {
      /** При каком % превышении цены продаем актив, чтобы зафиксировать прибыль
      takeProfit: 10,
      /** При каком % снижении цены продаем актив, чтобы не потерять еще больше
      stopLoss: 5,
    },
    /** Конфиг сигнала по скользящим средним */
    sma: {
      /** Кол-во точек для расчета быстрого тренда */
      fastLength: 10,
      /** Кол-во точек для расчета медленного тренда */
      slowLength: 30,
    },
```

Рисунок 2 - Функция настройки торгового робота

Это создает некоторую преграду для менее технически обученных пользователей, желающих воспользоваться системой, но не обладающих достаточными знаниями в области разработки программного обеспечения.

Кроме того, даже если пользователь обладает необходимыми навыками, структура подобных проектов может оказаться неудобной для модификации. Отсутствие визуального отображения процессов, а также невозможность оперативной настройки параметров без изменения исходного кода делает

использование таких систем тяжелым в понимании и подверженным ошибкам. Даже незначительные изменения могут повлечь некорректную работу программы, так как в данном случае не предусматриваются средства валидации вводимых данных, подсказки или защита от пользовательских ошибок.

Многие из существующих решений можно разбить на две категории. Такие решения либо предоставляют чрезмерно ограниченный набор функций, либо, наоборот, включают в себя излишне сложные и перегруженные модули. В первом случае такие системы не способны удовлетворить базовые потребности трейдера, так как выполняют лишь узкий спектр задач, чаще всего — запуск и остановку торгового робота. Во втором случае речь идёт о программных продуктах, в которых реализовано множество различных функций, однако их интерфейс обычно перегружен, плохо структурирован и ориентирован на профессиональных пользователей. Такие приложения дополнительно требуют значительного времени для изучения и настройки.

Пример реализации интерфейса такого многофункционального приложения можно увидеть на рисунке 3.

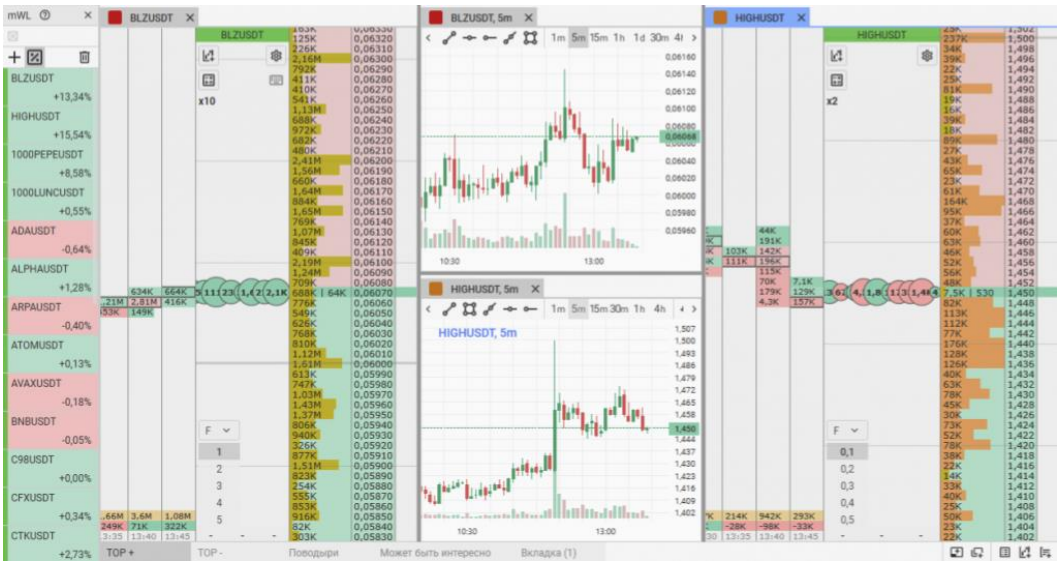

Рисунок 3 - Пример интерфейса программы для торговли на бирже

Особого внимания заслуживают коммерческие системы, которые обладают широкими возможностями, имеют проработанный интерфейс и обеспечивают постоянную техническую поддержку. Однако в таких решениях

отсутствует доступ к исходному коду. Пользователь лишен возможности адаптировать программу под свои индивидуальные потребности, проследить алгоритмы работы системы или удостовериться в корректности ее функционирования. Это создает ситуацию зависимости от разработчиков, а также снижает уровень доверия к таким продуктам.

На фоне вышеперечисленных недостатков текущая система занимает промежуточное положение между простыми скриптами для запуска торговых ботов и крупными профессиональными платформами. Она включает в себя открытость архитектуры, удобство применения и расширенную функциональность. Приложение может рассматриваться как универсальный помощник трейдера, предоставляющий доступ как к торговым алгоритмам, так и ко всем необходимым вспомогательным сопутствующим функциям.

Основной недостаток и в то же время преимущество данной системы – это ее промежуточное положение. Данная система может подойти далеко не каждому пользователю, так как она не может в полной мере конкурировать с полноценными платформами, интегрированными с большим числом брокеров, обладающими развитой системой аналитики и многопоточной архитектурой. В то же время она превосходит простые решения в виде скриптов, но это тоже необходимо далеко не каждому пользователю.

5. Архитектура системы

Структура проекта построена на принципах модульности, что позволяет разделить различные аспекты функциональности на независимые компоненты. Это упрощает управление проектом, его дополнение новыми функциями и интеграцию с внешними системами. Основные папки и файлы организованы по их функциональному назначению, что можно увидеть в структуре самого проекта:

– models. В данной папке находятся файлы, которые отвечают за создание сущностей базы данных;

– schemas. В данной папке находятся файлы, отвечающие за схемы данных, которые обеспечивают валидацию и сериализацию данных;

– backend/api. В данной папке находятся файлы, которые содержат в себе функции для взаимодействия с базой данных;

– client/api. В данной папке находятся файлы, которые взаимодействуют с маршрутами на backend-части приложения;

– bot. Данная папка содержит файл с экземпляром класса бота;

– config. Данная папка содержит в себе файлы, отвечающие за конфигурацию базы данных и планировщиков задач;

– graphics. Данная папка содержит в себе файлы, отвечающие за создание графиков;

– handlers. Данная папка содержит в себе обработчики команд каждого отдельного модуля;

– log. Данная папка содержит в себе файл с логами системы;

– orders. Данная папка содержит в себе файлы, отвечающие за торговые поручения. Пример программного кода для создания торгового поручения на покупку представлен в приложении 1;

– signals. Данная папка содержит в себе файлы, отвечающие за различные сигналы;

– store. Данная папка содержит файлы для временного хранения данных;

– strategy. Данная папка содержит в себе файлы, отвечающие за управление торговой стратегией;

– utils. Данная папка содержит в себе файлы, которые предоставляют вспомогательные функции;

– Dockerfile и docker-compose.yml. Файлы для создания и запуска docker-образа.

Для хранения информации была разработана база данных. Структура сущностей в базе данных показана на рисунке 4.

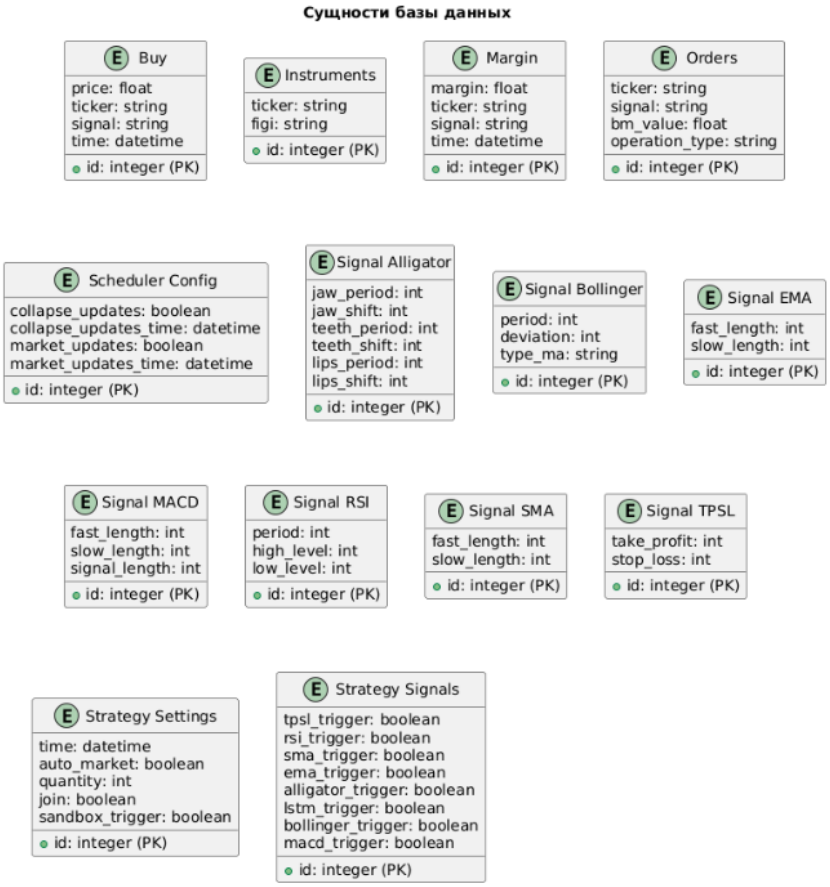

Рисунок 4 - Структура сущностей в базе данных

База данных состоит из множества сущностей, каждая из которых отвечает за определённую часть логики работы системы:

– Buy — таблица для хранения информации о покупках ценных бумаг. Содержит идентификатор операции (id), цену покупки (price), тикер бумаги (ticker), торговый сигнал или группу торговых сигналов,

инициировавших покупку (signal), а также дату и время совершения операции (time). Эти данные используются для анализа эффективности определенной стратегии, а также для последующего расчёта прибыли и убытков;

– Margin — таблица, идентичная Buy. Хранит в себе информацию о продажах ценных бумаг. Содержит идентификатор операции (id), значение маржи в процентах (margin), тикер бумаги (ticker), торговый сигнал или группу торговых сигналов, инициировавших покупку (signal), а также дату и время совершения операции (time);

– Orders — таблица для хранения промежуточных данных о выставленных торговых заявках. Содержит в себе поля для хранения тикера ценной бумаги (ticker), значения сигнала или группы сигналов (signal), цену покупки или продажи (bm_value) и тип операции (operation_type). Данная таблица необходима для отслеживания состояния ценной бумаги при покупке или продаже, чтобы корректно инициализировать записи в таблицах Buy или Margin;

– Instruments — служебная таблица, содержащая информацию об инструментах, добавленных пользователем в систему. Каждый инструмент описывается тикером (ticker) и FIGI (уникальным идентификатором инструмента в системе брокера);

– Scheduler Config — таблица для хранения настроек уведомлений и задач, выполняемых по расписанию. В таблице задаются флаги и временные метки для различных типов обновлений. Флагами служат поля collapse_updates и market_updates, а временные значения записываются в collapse_updates_time и market_updates_time;

– Strategy Settings — таблица, в которой хранятся пользовательские параметры выбранной стратегии. Данная таблица содержит поля для включения режима автоматической торговли (auto_market), хранения количества покупки и продажи ценных бумаг (quantity), режима песочницы (sandbox_trigger) и времени срабатывания стратегии. Все эти поля позволяют

пользователю легко настроить стратегию конкретно под себя и свои потребности;

– Strategy Signals — определяет, какие именно сигналы технического анализа будут использоваться в текущей торговой стратегии. Таблица содержит логические для каждого сигнала. Это даёт возможность включать или выключать отдельные индикаторы без изменения общего механизма работы стратегии.

Сигналы представлены отдельными таблицами, каждая из которых отвечает за параметры конкретного технического индикатора. Эти параметры используются при расчётах в торговой стратегии, генерации торговых сигналов и визуализации на графиках:

– Signal RSI — задаёт параметры для расчёта индикатора относительной силы. Включает в себя период скользящего окна (period), верхний и нижний уровни (high_level и low_level), которые сигнализируют о перекупленности или перепроданности конкретной ценной бумаги;

– Signal SMA и Signal EMA — хранят параметры для простой и экспоненциальной скользящих средних соответственно. Указываются значения для «быстрой» и «медленной» средней (fast_length и slow_length), используемых при анализе пересечений и определения тренда;

– Signal MACD — таблица с параметрами для индикатора MACD. Данная таблица включает в себя длины периодов для быстрой и медленной средних (fast_length и slow_length) и сигнальной линии (signal_length);

– Signal Bollinger — хранит настройки для полос Боллинджера, включая период (period), стандартное отклонение (deviation) и тип скользящей средней (type_ma), которая используется в центре полос;

– Signal Alligator — задаёт периоды и смещения для трёх линий индикатора Аллигатор (челюсти, зубы и губы), основанных на скользящих средних с разными параметрами. Эти линии используются для определения фаз тренда и консолидации;

– Signal TPSL — используется для установки уровней получения прибыли и остановки убытков (take_profit и stop_loss). Данный сигнал позволяет ограничивать убытки и фиксировать прибыль по каждой сделке.

6. Функциональные компоненты

Функциональные компоненты системы предназначены для автоматизации процессов управления инвестициями и упрощения взаимодействия пользователя с необходимыми данными. Каждая компонента реализует определенную задачу, обеспечивая эффективное взаимодействие с API и базой данных, а также предоставляя информацию в удобном и понятном формате. В их разработке использовались обработчики команд, методы интеграции с внешним API и таблицы базы данных, структурированные в соответствии с требованиями системы.

6.1. Портфолио

Для реализации получения текущего портфолио пользователя был создан обработчик portfolio. Основная логика его работы заключается в том, что данный обработчик получает в параметрах токен и тип счета: «боевой» или «песочница». В зависимости от выбранного типа вызывается метод получения портфолио из API Т-Инвестиций. Для каждой позиции в портфолио формируется читаемый текст, который отправляется пользователю.

Общую схему взаимодействия пользователя с модулем «Портфолио» можно увидеть на рисунке 5.

Рисунок 5 - Схема взаимодействия с модулем портфолио

6.2. Инструменты

Для реализации функционала инструментов был создан обработчик instruments и соответствующая таблица в базе данных. Данная компонента включает в себя четыре основных действия:

– добавить инструмент. Данная команда реализует добавление инструмента в базу данных. Пользователь вводит тикер инструмента, после чего ищется его FIGI-идентификатор. Если он найдет и данный тикер существует, то инструмент записывается в базу данных;

– получить мои инструменты. Команда, которая вызывает метод для получения всех инструментов из базы данных и отправляет их пользователю;

– удалить мои инструменты. Команда, которая вызывает метод удаления всех инструментов из базы данных;

– удалить инструмент. При выборе данной команды пользователю сначала присылается интерактивный список его инструментов. При нажатии на конкретный инструмент он будет удален из базы данных посредством вызова нужного метода.

Схему работы пользователя с функциональным модулем инструментов более подробно можно увидеть на рисунке 6.

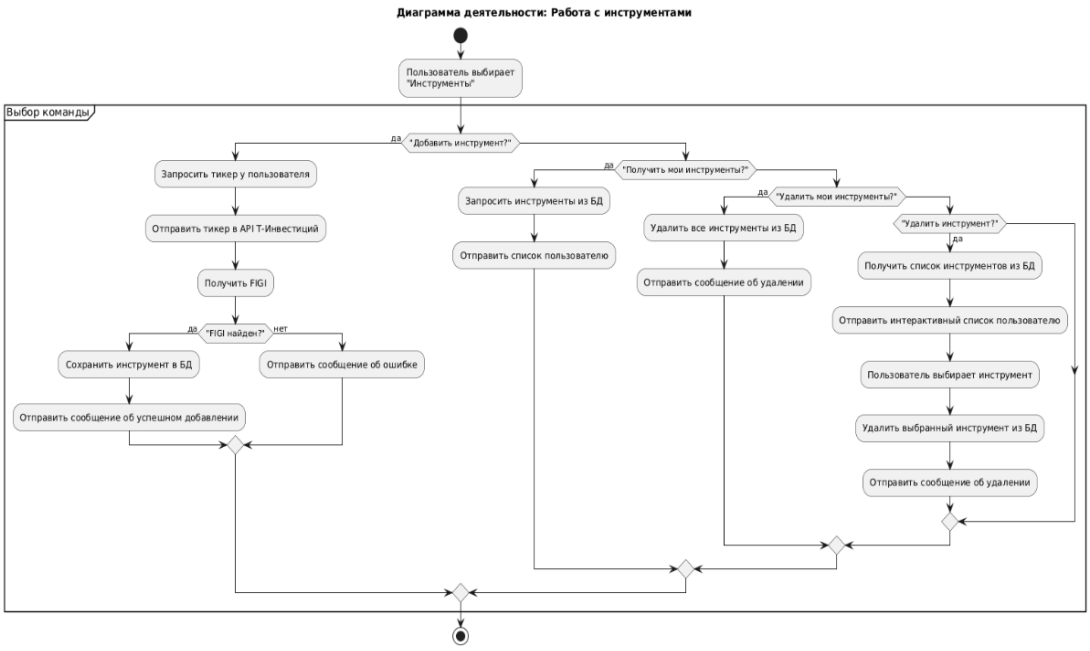

Рисунок 6 - Диаграмма деятельности для работы с инструментами

Для взаимодействия с инструментами на backend-части приложения был создан отдельный файл instruments.py, содержащий в себе маршруты для работы с инструментами. На клиентской части приложения также был создан файл instruments_client.py, содержащий в себе API для взаимодействия с серверной частью приложения, отвечающей за инструменты.

Программный код для добавления инструмента в систему представлен в приложении 2 и в приложении 3.

6.3. Уведомления

Для реализации данной компоненты был создан обработчик notification и соответствующая таблица Scheduler Config в базе данных. При проверке изменения цены вызывается специальный метод, который получает ценовые свечи инструмента на данный момент и введенное пользователем время назад, после чего сравнивает их значения. Программный код функции изменения цены представлен в приложении 4. У пользователя есть четыре доступных команды:

– подписка на падения рынка. При выборе данной команды пользователю предложат записать временной промежуток, через который ему будут отправляться уведомления. После указания промежутка нужные данные добавляются в соответствующую таблицу. При перезапуске приложения конфигуратор автоматически настроит планировщики задач по указанным пользователям параметрам. Данные уведомления будут присылать данные по инструментам пользователя, которые за указанный промежуток времени потеряли в стоимости какой-то процент;

– отписка от падений рынка. При выборе данной команды пользователь отпишется от соответствующих уведомлений, триггер в базе данных изменит свое значение на нулевое, а время удалится;

– подписка на обновления рынка. При выборе данной команды пользователю предложат записать временной промежуток, через который ему будут отправляться уведомления. После указания промежутка нужные данные добавляются в соответствующую таблицу. При перезапуске приложения

конфигуратор автоматически настроит планировщики задач по указанным пользователям параметрам. Данные уведомления будут присылать данные по инструментам пользователя, которые обновили свои показатели в какую-либо сторону;

– отписка от обновлений рынка. При выборе данной команды пользователь отпишется от соответствующих уведомлений, триггер в базе данных изменит свое значение на нулевое, а время удалится.

Схему работы пользователя с функциональным модулем уведомлений можно увидеть на рисунке 7.

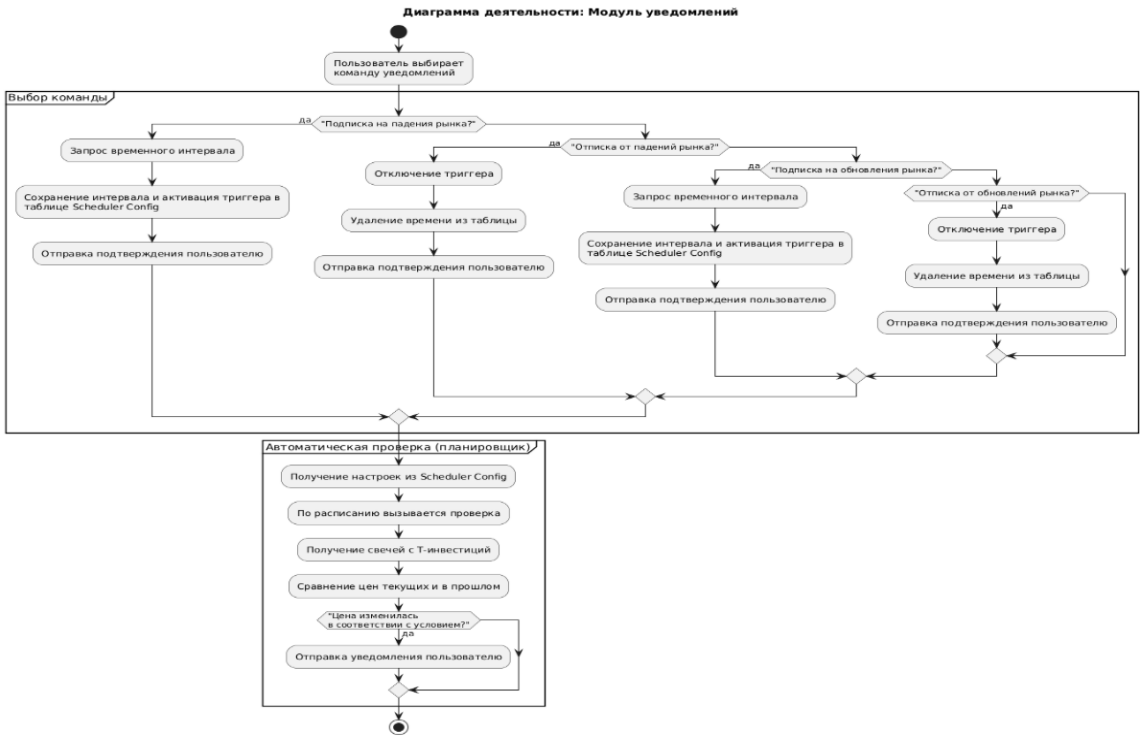

Рисунок 7 - Диаграмма деятельности для работы с уведомлениями

Для взаимодействия с уведомлениями на backend-части приложения был создан файл config.py, содержащий в себе маршруты для работы с конфигуратором уведомлений. На клиентской части приложения также был создан файл config_client.py, содержащий в себе API для взаимодействия с серверной частью приложения, отвечающей за уведомления.

6.4. Состояние рынка

Для реализации компоненты получения состояния рынка был создан обработчик market. Эта компонента предлагает пользователю аналогичные уведомлениям данные, только в текущий момент времени.

Данный обработчик содержит в себе три доступные команды:

– получить обвал рынка. При выборе пользователем данной команды ему предлагается сначала выбрать промежуток времени, за который необходимо получить свечи, а затем процент падения, на который должна была обвалиться ценная бумага. После этого пользователю отправятся обработанные и отформатированные данные в удобном для восприятия формате;

– получить рост рынка. Данная команда имеет такую же концепцию, как и обвал рынка, только указывается процент роста, а не падения;

– получить изменение состояния рынка. При выборе данной команды пользователь получает данные о всех движениях цен инструментов из его базы данных за введенный период.

Диаграмма деятельности для рассматриваемого модуля состояния рынка представлена на рисунке 8.

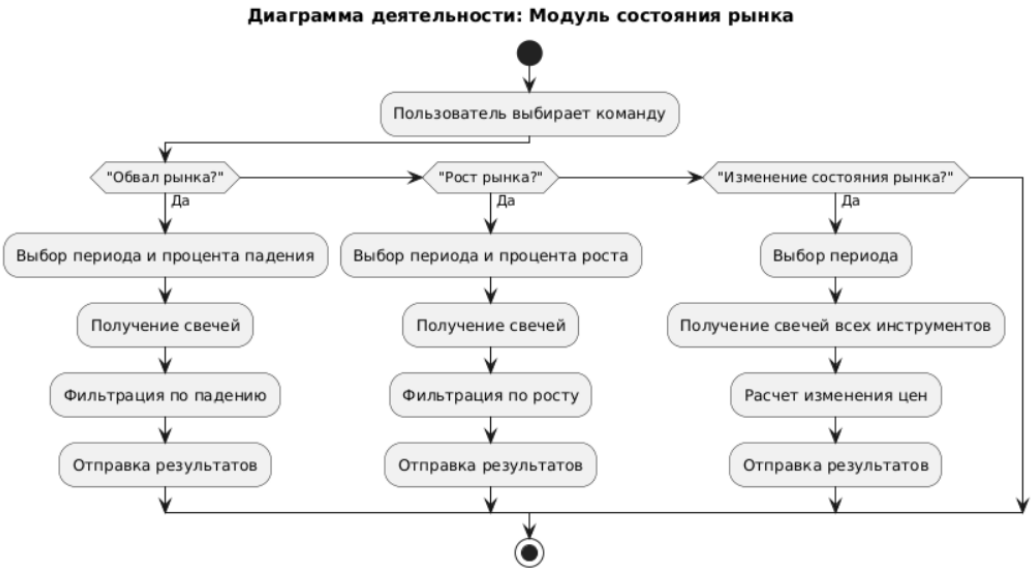

Рисунок 8 - Диаграмма деятельности для модуля состояния рынка

6.5. Настройка сигналов

Для данной компоненты был реализован обработчик signals и таблицы, соответствующие каждому определенному сигналу. При выборе данной команды пользователю присылается список всех доступных сигналов. После того, как пользователь выбрал определенный сигнал, ему будут присылаться сообщения с просьбой ввести значения для каждого необходимого поля сигнала. Какие значения вводить – пользователь выбирает сам. От введенных значений будет зависеть обработка сигналов и торговая стратегия. Для каждого типа значений были реализованы специальные валидаторы, не дающие ввести некорректные данные.

Диаграмма деятельности для модуля настройки сигналов представлена на рисунке 9.

Рисунок 9 - Диаграмма деятельности для модуля настройки сигналов

6.6. Торговый робот

Для реализации данной компоненты был создан обработчик bot и таблицы Strategy Signals и Strategy Settings. Стратегия срабатывает для каждого инструмента из базы данных пользователя через определенный

промежуток времени, который так же указывается пользователем. Для каждого инструмента проверяется каждый включенный в стратегию сигнал и проверяется возвращаемое значение сигнала. После проверки всех условий определяется требуемая операция, то есть покупка, продажа или удержание.

Всего пользователю доступны четыре команды:

– настроить стратегию. При выборе данной команды пользователю отправится список всех доступных сигналов. Пользователь может включить в стратегию любой сигнал, выбрав его, при условии, что данный сигнал был настроен. После выбора сигналов пользователь получит список временных промежутков, через которые необходимо, чтобы стратегия срабатывала. После этого пользователь должен выбрать, включить ли автоматическую торговлю или только присылать рекомендации по покупке, продаже или удержанию ценной бумаги. Далее пользователь должен будет указать количество ценных бумаг, которые будут продаваться и покупаться, а также выбрать, совершать торговую операцию при срабатывании хотя бы одного сигнала или всех сигналов сразу;

– отключить стратегию. После выбора пользователем данной команды торговая стратегия будет отключена;

– выбор счета. При выборе пользователем данной команды он может поменять счет, на котором будут совершаться расчеты и торговые операции. Если пользователь выберет «боевой» счет, то все торговые поручения будут выставляться на его основном счете с его действительным балансом, если же будет выбрана «песочница», то торговые операции будут производиться на счете с ненастоящими денежными средствами, но по реальным рыночным данным. Это может быть полезно, если пользователь хочет проверить свою стратегию в действии;

– информация о «песочнице». При выборе данной команды пользователь может получить портфолио счета своей «песочницы», либо пополнить баланс.

Весь алгоритм обработки пользовательской стратегии можно описать следующим образом:

1) загрузка переменных окружения;

2) получение токена и флага песочницы;

3) получение всех инструментов пользователя. Если у пользователя нет инструментов, сделать return и сообщить пользователю об этом;

4) получение конфигурации стратегии. Если конфигурация пустая, сделать return и сообщить об этом пользователю;

5) для каждого инструмента получить его FIGI-идентификатор;

6) получение текущей прибыли для актива;

7) получение сигналов для каждого актива на покупку, продажу или удержание;

8) проверка условия логического оператора срабатывания стратегии;

9) проверка условия на автоматическую торговлю. Если выбрана автоматическая торговля, то происходит завершение торговой заявки по данному инструменту, если она уже была выставлена ранее и размещение новой торговой заявки. Если выбрана ручная торговля, для пользователя формируется рекомендация по текущему финансовому инструменту;

10) оповещение пользователя.

Схему работы стратегии можно увидеть на рисунке 10.

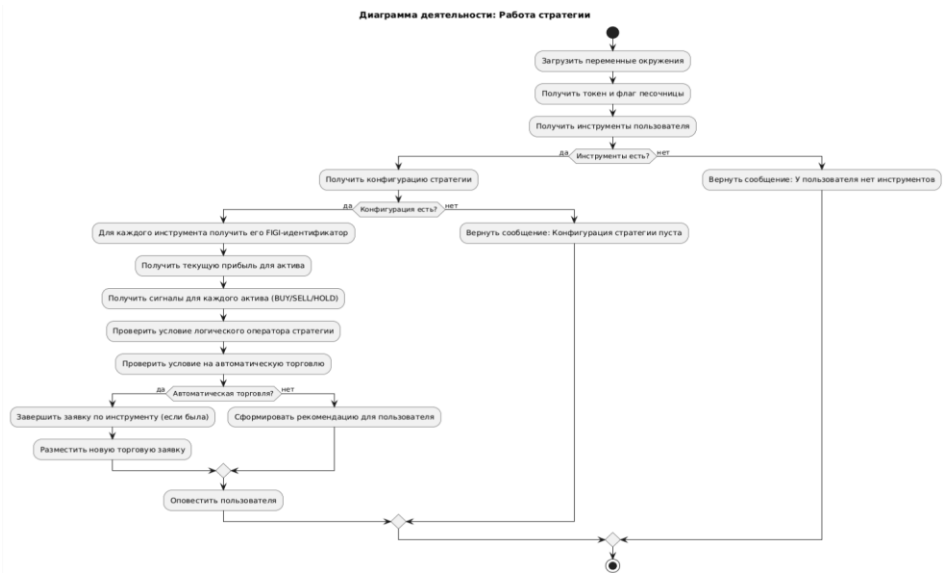

Рисунок 10 - Схема работы стратегии

При получении сигнала на покупку или продажу ценной бумаги стратегия вызывает функции из модуля orders. Данный модуль реализует функционал выставления торговых поручений. Схема работы компоненты для выставления торговых поручений представлена на рисунке 11.

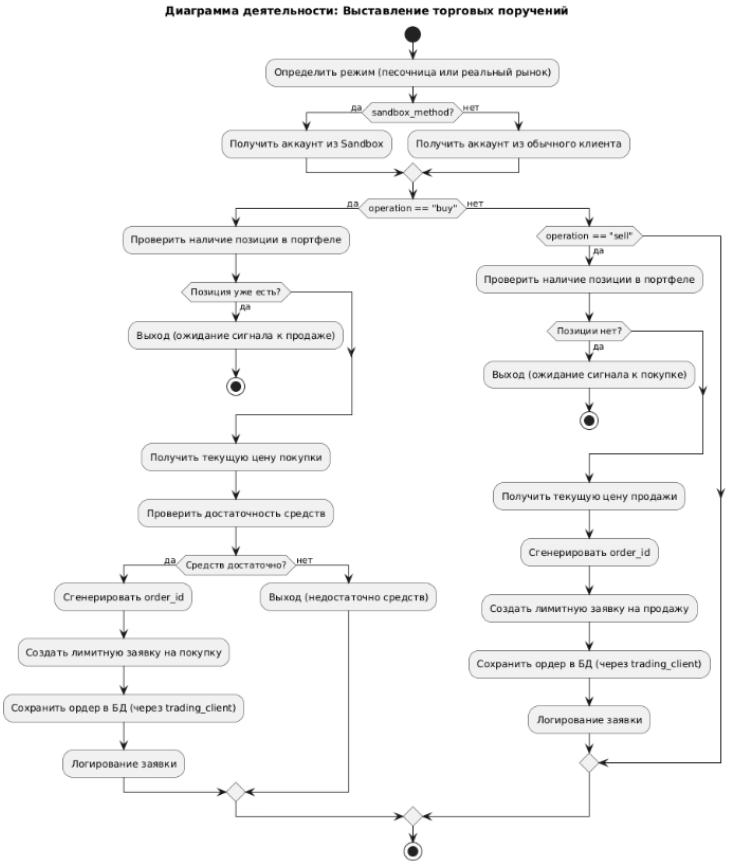

Рисунок 11 - Схема работы алгоритма выставления торговых поручений

6.7. Дивиденды и база знаний

Для компоненты дивидендов был реализован обработчик dividends. После выбора пользователем данной команды ему будет предложено указать период, за который он хочет посмотреть дивиденды по ценным бумагам, находящимся у него в базе данных. После введения периода специально разработанный метод из модуля utils вызовет для каждого инструмента функцию получения данных о дивидендах, сформирует отчет и отправит его пользователю.

Для компоненты базы знаний был создан обработчик knowledge_base. При выборе данной команды пользователю будет предложен список всех функциональных компонент. При выборе определенной функциональной

компоненты в базе знаний пользователю будет присылаться информация по данной компоненте. Например, пользователь может узнать, как настроить стратегию или определенный сигнал, получить рекомендации и базовые значения по их настройке или получить информацию по взаимодействию с определенным компонентом системы.

6.8. Статистика и графики

Для компоненты статистики был создан обработчик statistics и таблицы Buy и Margin. При выборе пользователем данной команды ему будет предложено либо ввести период, за который он желает получить статистику, либо получить общую статистику по покупкам и продажам. Если данных для отображения статистики нет, пользователю отправится соответствующее уведомление. Если же статистику рассчитать возможно, то пользователю отправятся графики количества покупок и продаж по определенным тикерам, по сигналам, по времени и соотношение положительной и отрицательной маржи, а также общая сумма покупок и маржи.

Схема взаимодействия пользователя с модулем статистики представлена на рисунке 12.

Рисунок 12 - Схема взаимодействия пользователя с модулем статистики

Для компоненты графиков был реализован обработчик graphics. Если пользователь выберет данную команду, ему будет предложено выбрать сигнал и промежуток времени для получения свечей. После получения ценовых

свечей будет построен специальный график, на котором будут изображены ценовые свечи и рассчитанные в модуле signals значения сигнала, выбранного пользователем.

7. Сигналы

Для работы стратегии или построения графиков необходимы некоторые индикаторы, которые в данной системе представлены в виде сигналов. Каждый сигнал имеет какие-то свои функциональные характеристики и особенности вычислений.

7.1. Сигнал SMA

SMA (Simple Moving Average) — представляет собой усреднённое значение цен закрытия за определённое количество периодов. Индикатор позволяет сгладить краткосрочные колебания цен и выявить общее направление движения актива.

Основные параметры сигнала:

– fast_length — длина (период) быстрой SMA, рассчитанной за меньшее количество последних цен;

– slow_length — длина (период) медленной SMA, рассчитываемой за большее количество последних цен;

– profit — текущая прибыль по активу, используется для принятия решения при продаже.

Данный сигнал показывает общее направление цены за заданный период времени. Пересечения между быстрой и медленной линиями SMA интерпретируются следующим образом:

– если быстрая SMA пересекает медленную снизу вверх — это сигнал на покупку;

– если быстрая SMA пересекает медленную сверху вниз и текущая прибыль положительная — это сигнал на продажу;

– во всех остальных случаях позиция сохраняется.

Алгоритм вычисления сигнала SMA в контексте системы выглядит следующим образом:

1) получение данных по свечам финансового инструмента, в частности цен закрытия;

2) расчёт быстрой и медленной скользящих средних с использованием библиотеки ta;

3) проверка условий пересечения двух линий. Если происходит пересечение снизу вверх — формируется сигнал на покупку финансового инструмента, если же происходит пересечение сверху вниз и текущая прибыль положительная — формируется сигнал на продажу финансового инструмента, во всех остальных случаях формируется сигнал на удержание позиции.

7.2. Сигнал EMA

EMA (Exponential Moving Average) — экспоненциальное скользящее среднее, которое придаёт больший вес более свежим значениям цен. EMA более чувствителен к последним изменениям рыночной цены по сравнению с простым скользящим средним. Индикатор используется для выявления направления тренда и определения потенциальных точек входа и выхода из позиции.

Основные параметры сигнала:

- fast_length — длина (период) быстрой EMA, рассчитанной по меньшему количеству последних цен закрытия;
- slow_length — длина (период) медленной EMA, рассчитанной по большему количеству последних цен закрытия;
- profit — текущая прибыль по активу, используется при формировании сигнала на продажу.

Данный сигнал отображает текущее направление цены, акцентируя внимание на последних изменениях. Пересечения между быстрой и медленной линиями EMA интерпретируются следующим образом:

- если быстрая EMA пересекает медленную снизу вверх — это сигнал на покупку;
- если быстрая EMA пересекает медленную сверху вниз и текущая прибыль положительная — это сигнал на продажу;
- в остальных случаях сигнал указывает на удержание текущей позиции.

Алгоритм вычисления сигнала EMA в контексте системы включает следующие этапы:

1) получение свечных данных по финансовому инструменту, в частности цен закрытия;

2) вычисление быстрой и медленной EMA при помощи библиотеки ta с заданными пользователем параметрами длины;

3) проверка условий пересечения двух линий. Если происходит пересечение снизу вверх — формируется сигнал на покупку финансового инструмента, если же происходит пересечение сверху вниз и текущая прибыль положительная — формируется сигнал на продажу финансового инструмента, во всех остальных случаях формируется сигнал на удержание позиции.

7.3. Сигнал RSI

RSI (Relative Strength Index) — это индекс относительной силы, измеряющий скорость и величину недавних изменений цен, чтобы оценить перекупленность или перепроданность актива. RSI принимает значения в диапазоне от 0 до 100 и помогает определить возможные точки разворота цены.

Основные параметры сигнала:

– period — период расчета RSI, определяет количество последних цен закрытия, на основе которых рассчитывается индикатор;

– low_level — нижний порог уровня RSI, используемый как сигнал перепроданности и потенциальной покупки;

– high_level — верхний порог RSI, используемый как сигнал перекупленности и потенциальной продажи;

– profit — текущая прибыль по активу, используется для фильтрации сигнала на продажу.

Индикатор RSI используется следующим образом:

– если RSI опускается ниже low_level — это сигнал на покупку (актив находится в зоне перепроданности);

– если RSI поднимается выше high_level и текущая прибыль положительна — это сигнал на продажу (актив в зоне перекупленности);

– во всех других случаях рекомендуется удержание позиции.

Алгоритм вычисления сигнала RSI в контексте системы состоит из следующих шагов:

1) получение исторических данных по свечам, в частности цен закрытия;

2) вычисление разностей между последовательными закрытиями;

3) расчёт приростов и потерь по каждой свече. Эти значения сглаживаются с помощью экспоненциального скользящего среднего;

4) определение относительной силы RS как отношения средней величины приростов к средней величине потерь;

5) вычисление индекса RSI по формуле $RSI = 100 - 100 / (1 + RS)$;

6) проверка индекса RSI. Если индекс меньше low_level, то генерируется сигнал на покупку, если же он больше high_level и прибыль положительна, генерируется сигнал на продажу, во всех остальных случаях формируется сигнал на удержание.

7.4. Сигнал MACD

MACD (Moving Average Convergence Divergence) — это индикатор импульса, основанный на разнице между двумя экспоненциальными скользящими средними с разными периодами. Он показывает, насколько быстро меняется цена, и помогает определить точки входа и выхода из позиции.

Основные параметры сигнала:

– fast_length — период для быстрой EMA;

– slow_length — период для медленной EMA;

– signal_length — период для сигнальной линии;

– profit — текущая прибыль по позиции, используется при проверке условий для продажи.

Алгоритм вычисления MACD:

1) получение исторических данных о ценах закрытия;

2) расчёт двух экспоненциальных скользящих средних. Быстрая EMA рассчитывается по короткому периоду, медленная по более длинному периоду;

3) разница между быстрой и медленной EMA образует основную линию MACD;

4) для сглаживания MACD рассчитывается сигнальная линия — EMA от значений MACD за период signal_length;

5) сигнал формируется на основе взаимного расположения линий. Если MACD выше сигнальной линии — это сигнал на покупку, если MACD ниже сигнальной линии и есть положительная прибыль — это сигнал на продажу, во всех остальных случаях — удержание позиции.

Программный код реализации торгового сигнала MACD представлен в приложении 5.

7.5. Сигнал Bollinger

Полосы Боллинджера — это индикатор волатильности, состоящий из трёх линий: скользящей средней, верхней и нижней полосы. Верхняя и нижняя полосы формируются на основе стандартного отклонения от средней, что позволяет оценивать перекупленность или перепроданность актива.

Основные параметры сигнала:

– period — период, за который рассчитываются скользящая средняя и стандартное отклонение;

– stddev — коэффициент стандартного отклонения, определяющий ширину полос;

– ma_type — тип скользящей средней;

– profit — текущая прибыль, используется для фильтрации сигнала на продажу.

Алгоритм расчёта полос Боллинджера:

1) получение цен закрытия за заданный период;

2) рассчитывается средняя линия. Если выбран тип SMA, используется простая скользящая средняя, если же выбран тип EMA, применяется экспоненциальная скользящая;

3) на основе отклонения цен от средней рассчитывается стандартное отклонение;

4) верхняя полоса рассчитывается по формуле $middle\_band + (rolling\_std * stddev)$;

5) нижняя полоса рассчитывается по формуле $middle\_band - (rolling\_std * stddev)$;

6) сигнал формируется на основе текущей цены. Если цена меньше или равна нижней полосы — это сигнал на покупку, если цена больше или равна верхней полосы и есть положительная прибыль — это сигнал на продажу, в остальных случаях — удержание позиции.

7.6. Сигнал Alligator

Alligator — это трендовый индикатор, основанный на трёх сглаженных скользящих средних, рассчитанных на основе средней цены бара, а не цены закрытия. Аллигатор помогает выявить фазы «сна» и «пробуждения» рынка, определяя начало и окончание тренда.

Основные параметры сигнала:

1) jaw_period — период сглаженной SMA для линии «челюсти», отражающей долгосрочную тенденцию;

2) teeth_period — период сглаженной SMA для линии «зубы», отражающей среднесрочную тенденцию;

3) lips_period — период сглаженной SMA для линии «губы», отражающей краткосрочную тенденцию;

4) jaw_shift, teeth_shift, lips_shift — сдвиги соответствующих линий вперёд на графике;

5) profit — текущая прибыль по активу, используется для принятия решения при продаже.

Индикатор интерпретируется на основе взаимного расположения линий «челюсти», «зубов» и «губ»:

– если губы пересекают зубы снизу вверх, а зубы пересекают челюсти снизу вверх — это сигнал на покупку;

– если губы пересекают зубы сверху вниз, а зубы пересекают челюсти сверху вниз и текущая прибыль положительная — это сигнал на продажу;

– все остальные варианты интерпретируются как сигнал на удержание позиции.

Алгоритм вычисления сигнала Alligator в контексте системы выглядит следующим образом:

1) получение минимумов (High) и максимумов (Low) по свечам финансового инструмента;

2) расчёт средней цены каждой свечи как $(High + Low) / 2$;

3) построение трёх скользящих средних на основе средней цены. «челюсти» вычисляются как SMA с наибольшим периодом и сдвигом, «зубы» вычисляются как средняя по периоду SMA, а «губы» — самая быстрая SMA;

4) применение сдвигов для каждой линии;

5) проверка условий пересечения.

Программный код реализации торгового сигнала Alligator представлен в приложении 6.

7.7. Сигнал LSTM

Данный сигнал основан на методе машинного обучения LSTM (Long Short-Term Memory). Данный метод машинного обучения основан на нейронной сети с долгой краткосрочной памятью. Он предназначен для анализа временных рядов и может учитывать как краткосрочные, так и долгосрочные зависимости в данных. В контексте данной системы, модель LSTM каждый раз обучается индивидуально на данных конкретного финансового инструмента. Это оправдано тем, что активы на бирже

значительно различаются по ценовым диапазонам и поведению, и обобщённое обучение на всех активах привело бы к крайне неточным предсказаниям.

Основные параметры сигнала:

– candles — исторические данные по свечам финансового инструмента, используемые для предобработки и генерации прогноза;

– profit — текущая прибыль по активу, используемая при принятии решения о продаже.

Алгоритм сигнала LSTM в контексте данной системы представлен следующим образом:

1) загрузка исторических данных. Модель обучается на истории цен конкретного инструмента, загружаемой через Tinkoff Invest API;

2) период делится на отрезки длиной не более одного года, после чего все данные объединяются в один DataFrame с датами и ценами закрытия;

3) масштабирование данных. Цены закрытия нормализуются с помощью MinMaxScaler в диапазон от 0 до 1;

4) формирование обучающей выборки. Каждой точке предсказания соответствует предыдущая последовательность из 60 значений, то есть модель обучается предсказывать цену, опираясь на паттерны длиной в 60 дней;

5) построение и обучение модели. Создаётся нейронная сеть с двумя слоями LSTM и двумя полносвязными слоями Dense. Модель обучается один раз, непосредственно перед предсказанием на текущем финансовом инструменте;

6) подготовка входа для прогноза. Берутся последние 60 цен закрытия, масштабируются тем же scaler, и преобразуются в формат трехмерного массива, состоящего из примеров, временных шагов и признаков;

7) прогноз следующей цены. Модель делает предсказание следующей цены. Результат масштабируется обратно в реальные значения;

8) генерация сигнала. Предсказанная цена сравнивается с текущей финансового инструмента. Если предсказанная цена выше текущей цены, то формируется сигнал на покупку, если предсказанная цена ниже текущей цены

и есть зафиксированная прибыль, то формируется сигнал на продажу, во всех остальных случаях формируется сигнал на удержание.

8. Развертывание системы

Развертывание системы возможно двумя способами: путем непосредственного запуска программы или с использованием контейнеризации через Docker. Оба варианта позволяют обеспечить работоспособность системы, но контейнеризация предоставляет дополнительное удобство в управлении зависимостями и изоляции окружения.

Перед тем, как осуществлять запуск системы, необходимо прописать значения для токенов счетов Т-Инвестиций, идентификатор чата в Telegram, а также токен бота.

Запуск системы осуществляется через основной исполняемый файл, логирование действий осуществляется с использованием встроенного модуля logging [6]. В данном файле выполняются несколько ключевых операций:

– настройка окружения и подключение всех необходимых зависимостей и конфигурационных файлов, а также загрузка переменных окружения;

– инициализация бота и API-клиента;

– инициализация и запуск сервера FastAPI. Данный сервер выполняется в отдельном потоке, чтобы не блокировать работу основного процесса;

– настройка базы данных и планировщиков задач;

– определение команд и кнопок, с помощью которых пользователь будет взаимодействовать с ботом;

– запуск основного процесса, в котором бот начинает работу в режиме polling, непрерывно ожидая сообщения от пользователя.

Пример терминала после успешного запуска бота с настроенными планировщиками уведомлений и неактивной стратегией можно увидеть на рисунке 13.

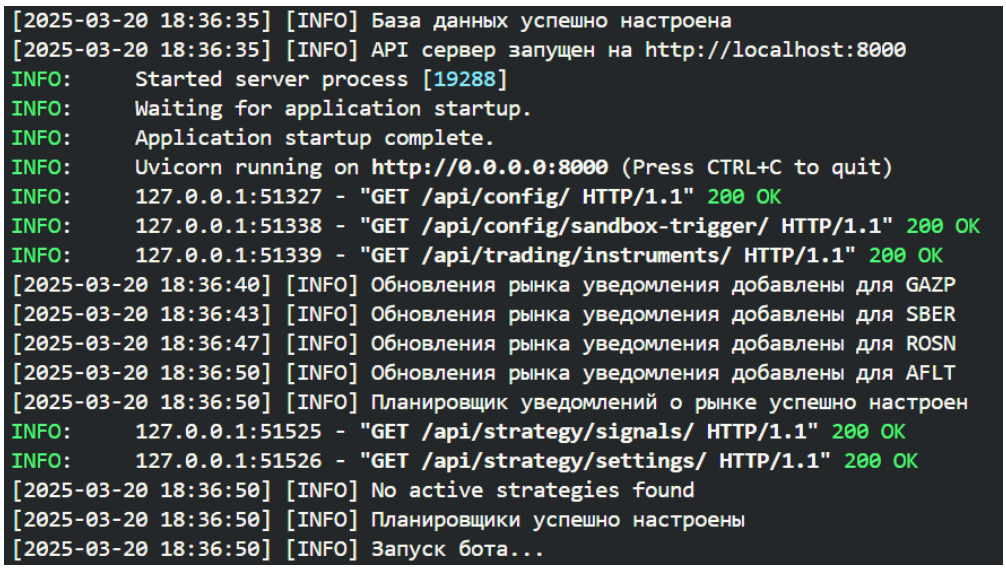

Рисунок 13 - Состояние терминала после успешного запуска приложения

9. Тестирование и эксплуатация системных модулей

Для проверки работоспособности приложения и его эксплуатационной пригодности будет проводиться ручное тестирование каждого отдельного модуля с подробным описанием действий. Также, где это возможно, будет происходить сравнение непосредственно с официальным сайтом брокера и данными, которые на нем отображаются.

9.1. Модуль портфолио

Функция получения портфолио выполняется при нажатии на кнопку «Получить портфолио». После нажатия на данную кнопку система отправляет запрос брокеру, после чего полученные данные форматируются в удобный для чтения вид и предоставляются пользователю. Пример полученного ответа можно увидеть на рисунке 14 и на рисунке 15.

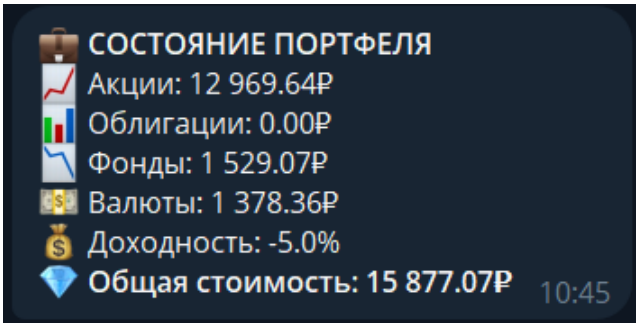

Рисунок 14 - Пример ответа о состоянии портфеля пользователя

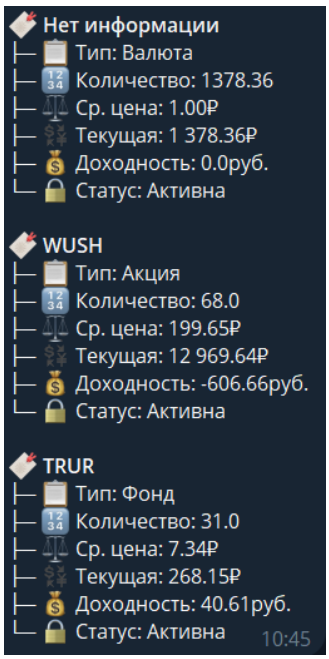

Рисунок 15 - Пример ответа о позициях в портфеле пользователя

Из полученных ответов пользователь может извлечь главную информацию о состоянии своего портфеля и активах, находящихся в нем.

Для портфеля пользователь может узнать информацию о количестве различных валютных инструментов, выраженную в рублях, а также доходность и общую стоимость активов.

Для каждого инструмента пользователь получает более детальную информацию, которая раскрывает следующие параметры инструмента:

– тип инструмента,
– количество в портфеле,
– средняя цена,
– цена на момент отправки отчета,
– доходность на момент отправки отчета,
– статус инструмента.

Все приходит в удобном для пользователя виде, с добавлением эмодзи для лучшей концентрации на отдельных пунктах. Можно заметить, что у некоторых инструментов отсутствует название и оно заменено на «Нет информации». Это происходит в виду того, что у некоторых биржевых инструментов, таких как валюты и фонды, может отсутствовать поле name, что учитывается приложением.

Для сравнения рассмотрим скриншот из приложения брокера, который представлен на рисунке 16.

Рисунок 16 - Скриншот из приложения брокера

Как можно заметить, данные в отчете и на скриншоте различаются с небольшой погрешностью. Это происходит в связи с постоянным обновлением данных в приложении брокера, в то время как данные в отчете актуальны на момент отправки отчета.

9.2. Модуль инструментов

Модуль инструментов включает в себя функциональности добавления, отображения и удаления. Визуальное представление управления инструментами пользователя можно увидеть на рисунке 17.

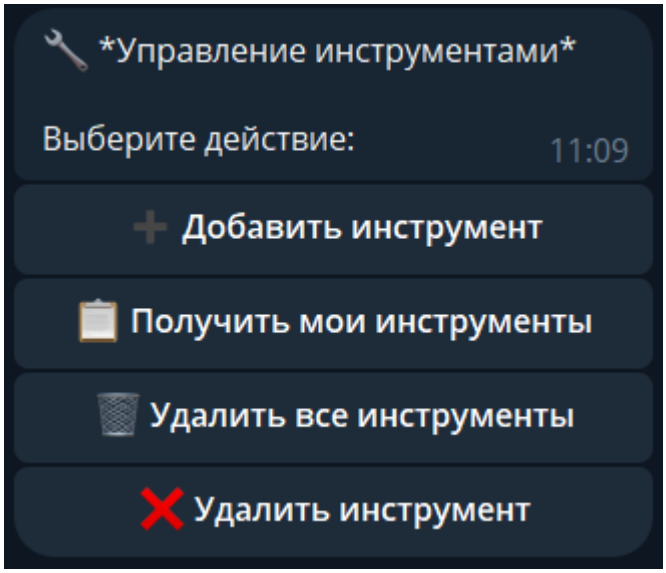

Рисунок 17 - Управление инструментами пользователя

Необходимо опробовать и протестировать каждую функциональность по-отдельности.

При нажатии на кнопку «Добавить инструмент» система предложит пользователю ввести тикер инструмента, как показано на рисунке 18.

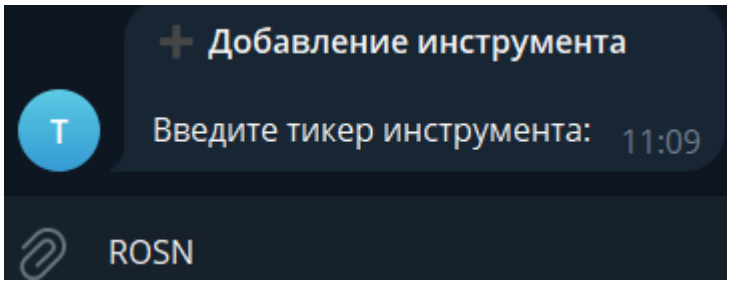

Рисунок 18 - Функциональность добавления инструмента

Если данный инструмент уже существует в базе данных, пользователю придет соответствующее уведомление, как показано на рисунке 19.

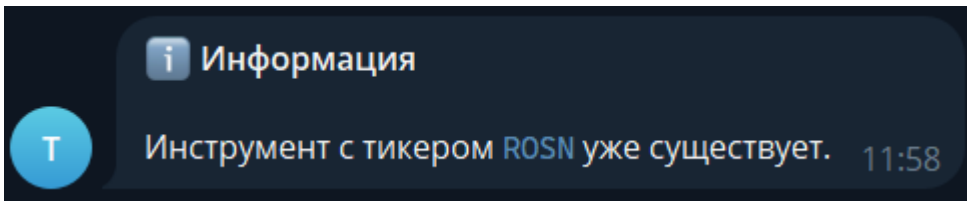

Рисунок 19 - Уведомление о существовании инструмента

Случай с некорректно введенным тикером также обрабатывается, это можно увидеть на рисунке 20.

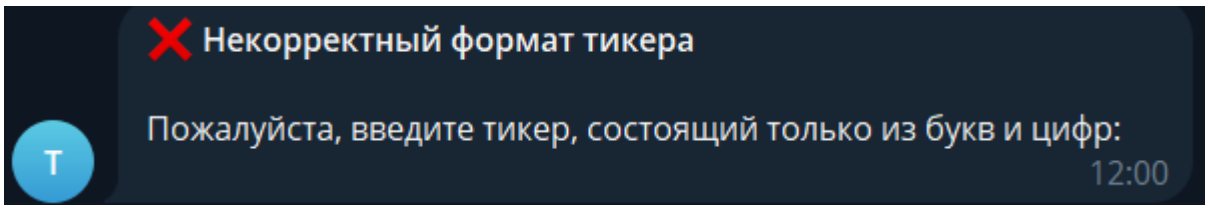

Рисунок 20 - Обработка некорректно введенного тикера

При вводе корректного тикера, если он отсутствует в базе данных, инструмент успешно добавится в систему и пользователь получит уведомление, которое можно увидеть на рисунке 21.

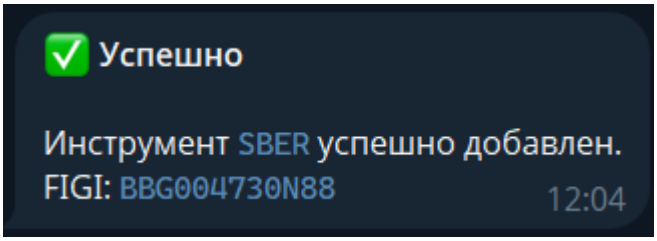

Рисунок 21 - Успешное добавление инструмента

При нажатии на кнопку «Получить мои инструменты» пользователю придет список его инструментов, который будет включать тикер каждого инструмента и его специальный идентификатор. Данный список можно увидеть на рисунке 22.

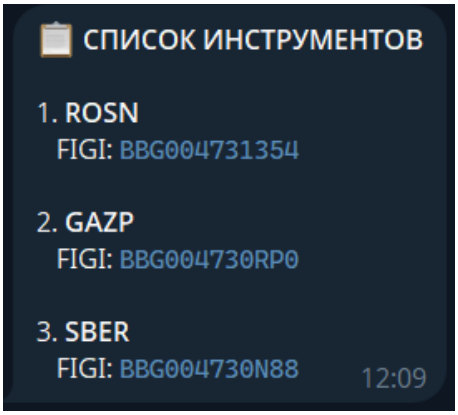

Рисунок 22 - Список инструментов пользователя

При отсутствии у пользователя добавленных инструментов, ему придет соответствующее уведомление, которое можно увидеть на рисунке 23.

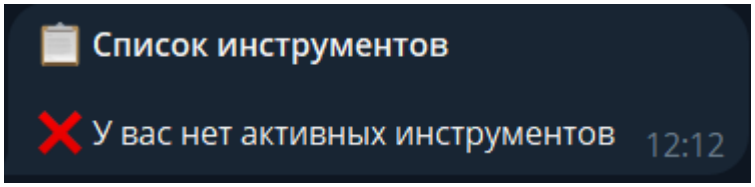

Рисунок 23 - Уведомление об отсутствии инструментов

При нажатии на кнопку «Удалить инструмент» пользователю придет список его инструментов в виде активных кнопок. Данный список представлен на рисунке 24.

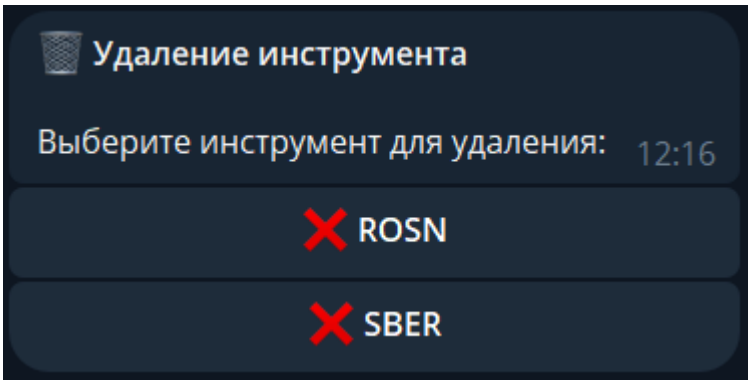

Рисунок 24 - Список инструментов для удаления

После выбора соответствующего инструмента, он удалится из базы данных и пользователю придет соответствующее уведомление, которое показано на рисунке 25. Программный код модуля для удаления инструмента представлен в приложении 7.

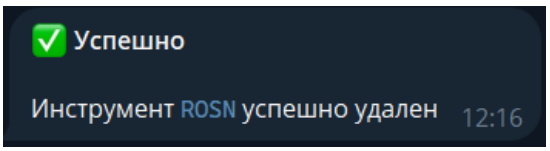

Рисунок 25 - Уведомление об удалении инструмента

При нажатии на кнопку «Удалить все инструменты» пользователю придет уведомление об удалении всех его добавленных инструментов из базы данных, что можно увидеть на рисунке 26.

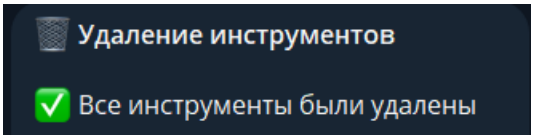

Рисунок 26 - Уведомление об удалении всех инструментов

9.3. Модуль уведомлений

Модуль уведомлений представлен в виде четырех функциональных команд, которые можно увидеть на рисунке 27.

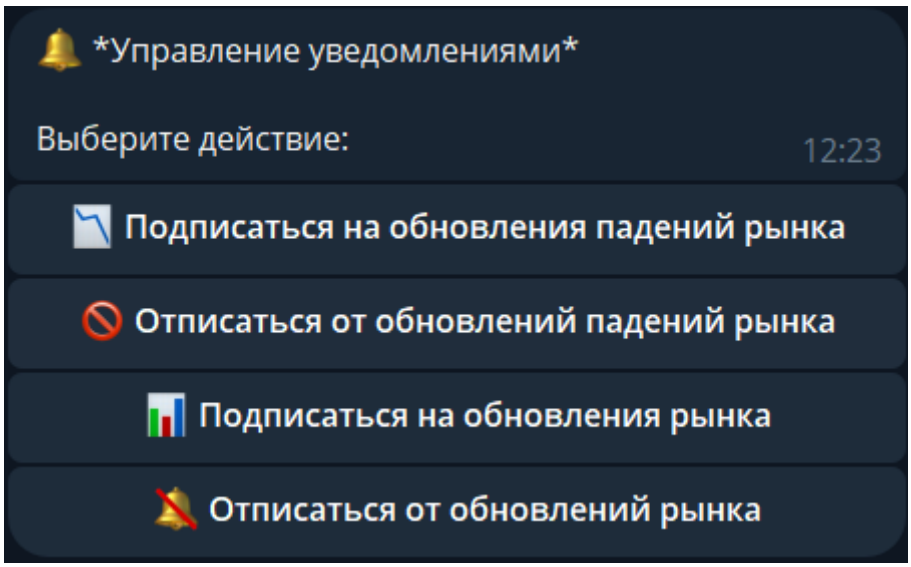

Рисунок 27 - Модуль управления уведомлениями

При выборе команды «Подписаться на обновления падений рынка» или команды «Подписаться на обновления рынка» система предложит пользователю выбрать интервал для получения уведомлений, как показано на рисунке 28.

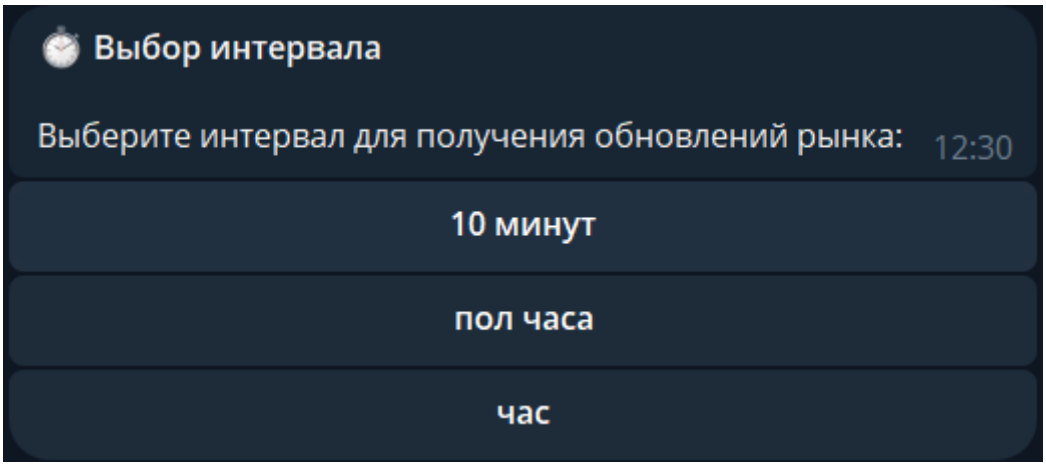

Рисунок 28 - Выбор интервала уведомлений

После выбора интервала уведомлений система настраивает планировщики для каждого инструмента и присылает уведомление об успешной подписке, что можно увидеть на рисунке 29 и на рисунке 30.

```
[2025-04-15 12:30:54] [INFO] Обновления рынка уведомления добавлены для WUSH
[2025-04-15 12:30:58] [INFO] Обновления рынка уведомления добавлены для GAZP
[2025-04-15 12:31:02] [INFO] Обновления рынка уведомления добавлены для ROSN
[2025-04-15 12:31:06] [INFO] Обновления рынка уведомления добавлены для SBER
[2025-04-15 12:31:06] [INFO] Планировщик уведомлений о рынке успешно настроен
```

Рисунок 29 - Логирование планировщиков уведомлений

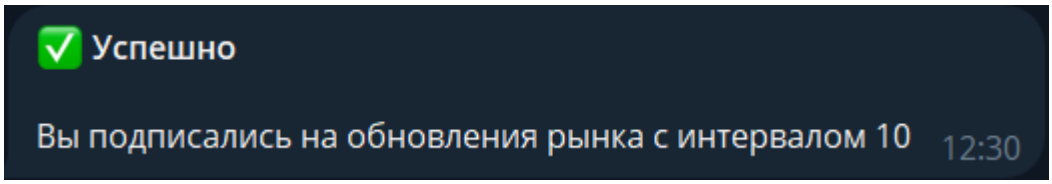

Рисунок 30 - Успешная подписка на уведомления

По прошествии указанного срока пользователь получит уведомления по каждой бумаге. Пример уведомления представлен на рисунке 31.

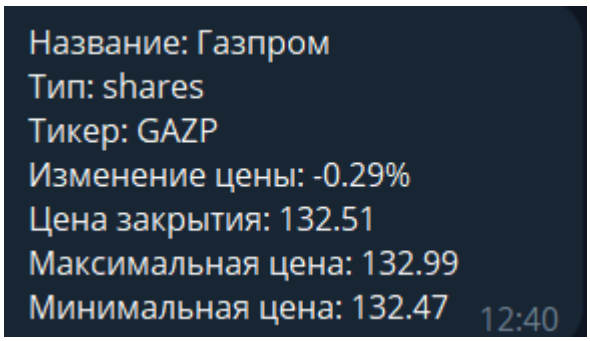

Рисунок 31 - Пример уведомления по изменению цены инструмента

Для каждого инструмента система специальным запросом получает ценовые свечи за выбранный интервал, после чего определяет изменение цены и предоставляет пользователю в удобном для восприятия виде. Пользователь получает информацию о следующих параметрах ценной бумаги:

– название,
– тип,
– тикер,
– изменение цены за выбранный период,
– цена закрытия,
– максимальная цена за выбранный период,
– минимальная цена за выбранный период.

Если сравнивать информацию из уведомления с терминалом, который представлен на рисунке 32, можно увидеть тенденцию на падение ценной

бумаги. Если говорить о процентном изменении цены, присутствует небольшая погрешность. Это происходит, так как в терминале данные обновляются каждые несколько секунд, а данные из уведомления актуальны на момент получения.

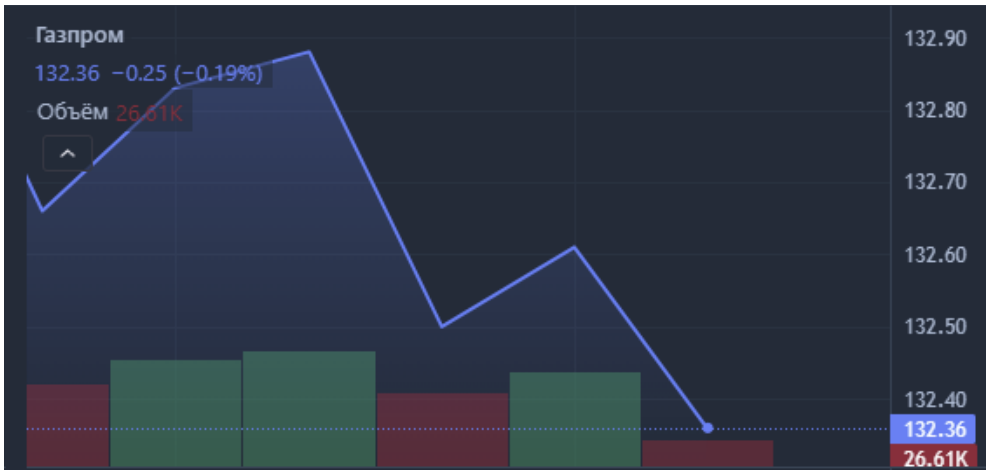

Рисунок 32 - Состояние терминала по ценной бумаге

При выборе пользователем команд «Отписаться от уведомлений» он будет отписан от соответствующих уведомлений и получит об этом предупреждение, которое представлено на рисунке 33.

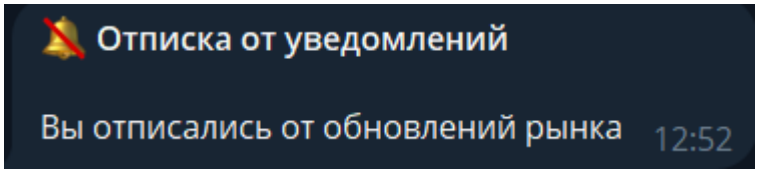

Рисунок 33 - Предупреждение об отписке от уведомлений

9.4. Модуль состояния рынка

Данный модуль предлагает пользователю три команды, которые представлены на рисунке 34.

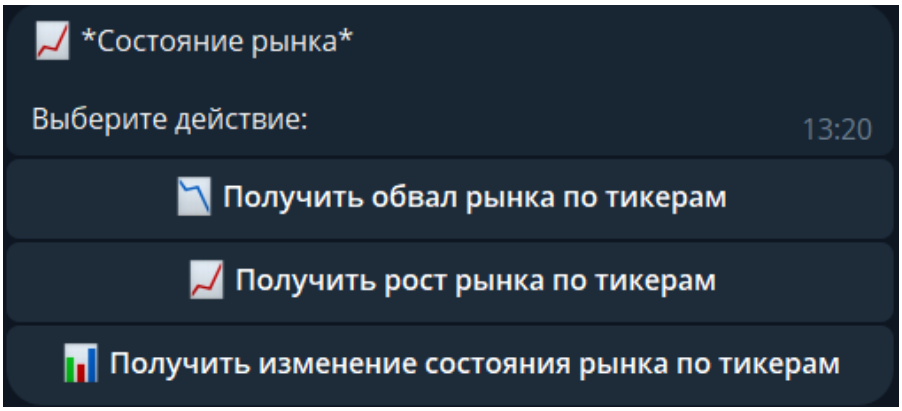

Рисунок 34 - Доступные пользователю команды

При выборе любой команды в данном модуле будет действовать примерно тот же механизм, что и у уведомлений. Для выбранного пользователем временного интервала, который представлен на рисунке 35, система будет получать ценовые свечи и вычислять изменение цены бумаги, после чего, в зависимости от выбранного условия изменения цены, присылать соответствующие уведомления. Пример уведомления об изменении цены можно увидеть на рисунке 36.

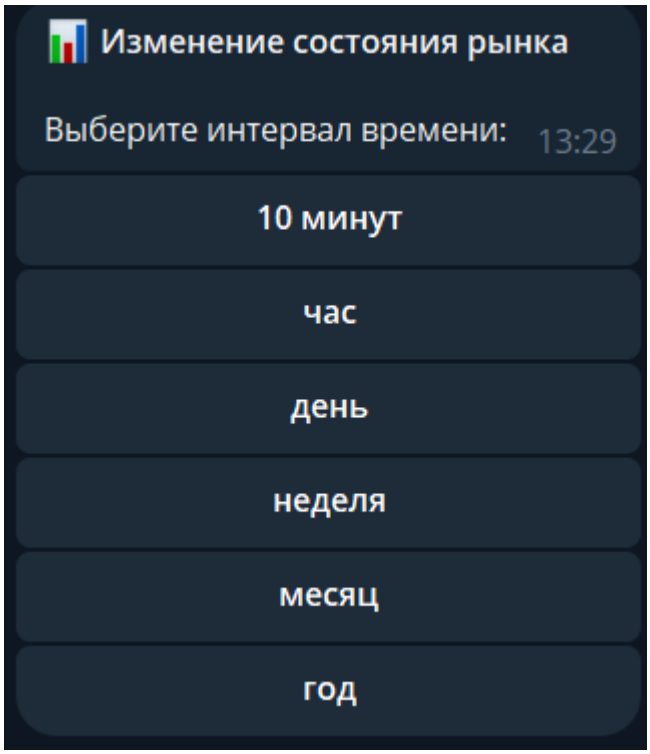

Рисунок 35 - Выбор интервала для получения информации о ценах

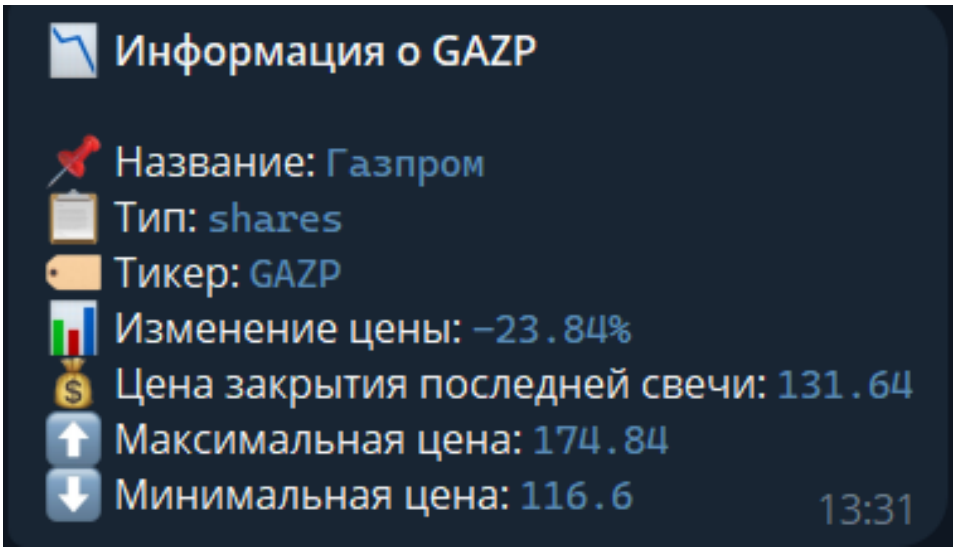

Рисунок 36 - Уведомление по ценной бумаге за выбранный период

В уведомлении пользователь получает следующую информацию:

– название ценной бумаги,
– тип,
– тикер,
– изменение цены за выбранный период,
– цена закрытия последней свечи,
– максимальная цена,
– минимальная цена.

Стоит уточнить, что корректно сравнить изменение цен в уведомлении и в терминале брокера не получится, так как система высчитывает изменение цены в интервале от месяца назад до текущего дня, в то время как в терминале информация предоставляется в интервале от начала прошлого месяца до начала текущего месяца.

9.5. Модуль настройки сигналов

Модуль настройки сигналов предоставляет пользователю настроить любой сигнал, который предусмотрен системой. Для любого сигнала пользователь может по своему желанию указать абсолютно любые значения для каждого параметра. При нажатии на кнопку «Настройка сигналов» пользователь получит список сигналов в виде функциональных кнопок, которые представлены на рисунке 37.

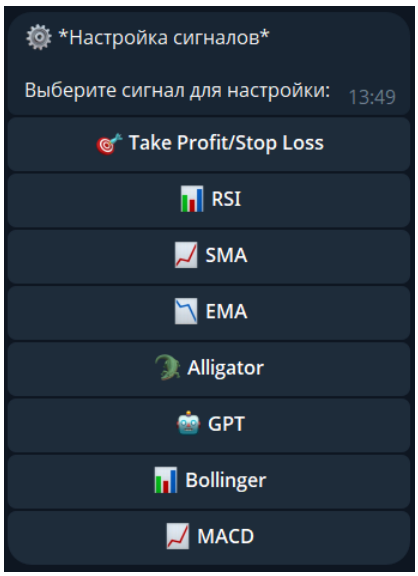

Рисунок 37 - Список сигналов для настройки

В качестве примера будет рассмотрена настройка сигнала SMA [7]. При выборе соответствующего сигнала пользователю будет предложено ввести значение первого параметра, а после и всех остальных параметров. Пример запроса на ввод параметра представлен на рисунке 38.

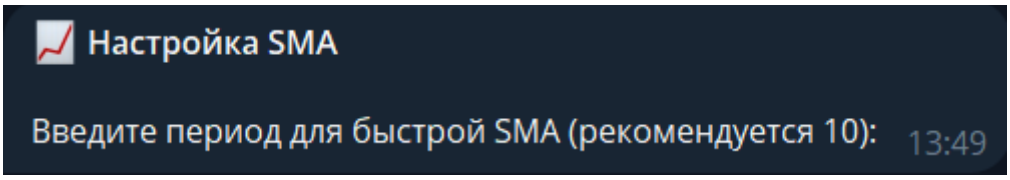

Рисунок 38 - Запрос системы на ввод параметра для быстрой SMA

После ввода параметра быстрой SMA система предложит ввести параметр медленной SMA, данное действие представлено на рисунке 39.

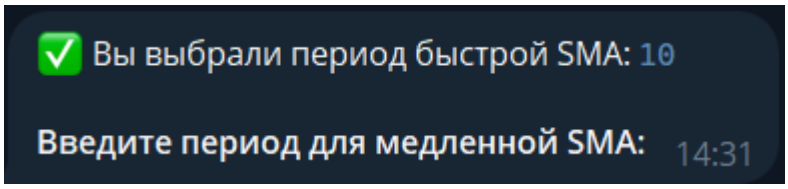

Рисунок 39 - Запрос системы на ввод параметра для медленной SMA

После ввода всех параметров, система отправит пользователю соответствующее уведомление, что представлено на рисунке 40, и обновит соответствующие параметры сигнала в базе данных, что можно увидеть на рисунке 41.

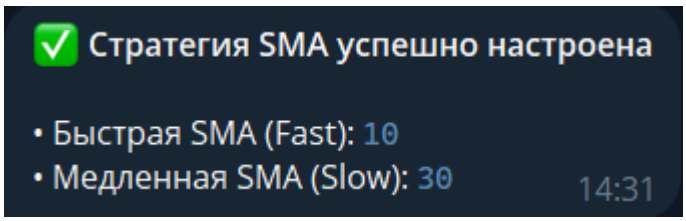

Рисунок 40 - Успешная настройка сигнала SMA

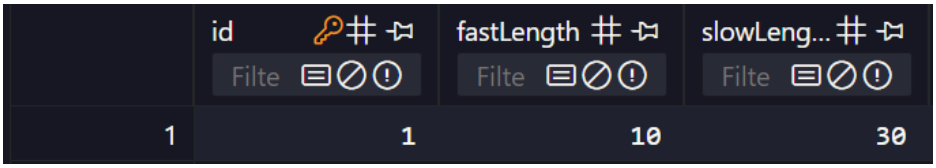

Рисунок 41 - Параметры сигнала SMA в базе данных

Пример программного кода для настройки сигнала Alligator представлен в приложении 8.

9.6. Модуль торгового робота

Модуль торгового робота предоставляет пользователю функционал для торговли на бирже. Доступный пользователю набор команд данного модуля представлен на рисунке 42.

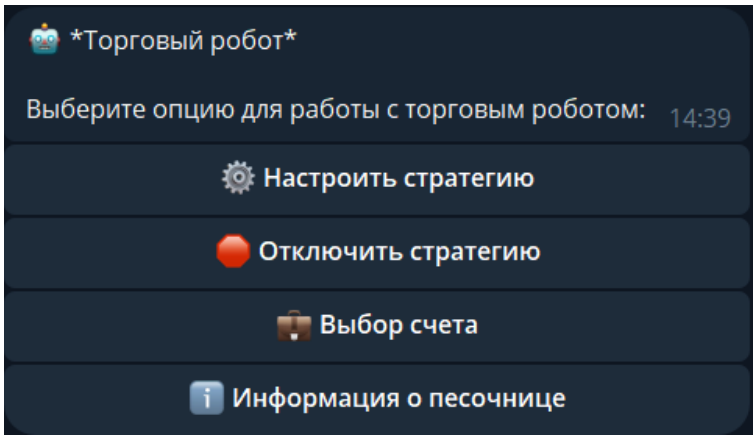

Рисунок 42 - Команды модуля торгового робота

При выборе команды «Настроить стратегию» пользователь перейдет в режим настройки стратегии. Первым делом пользователю придет список функциональных кнопок с выбором соответствующих сигналов, который представлен на рисунке 43. При выборе конкретного сигнала он добавляется во временный список хранения сигналов. Если же был выбран сигнал, который еще не настроен, пользователю отправится соответствующее предупреждение, которое можно увидеть на рисунке 44.

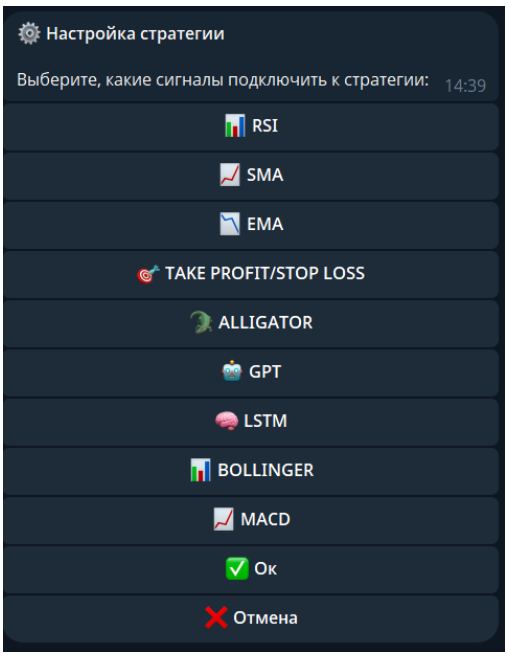

Рисунок 43 - Выбор сигналов для стратегии

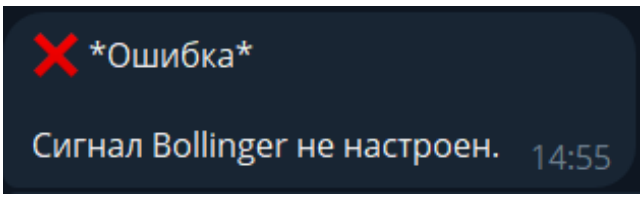

Рисунок 44 - Уведомление о настройке сигнала

После выбора всех нужных сигналов и нажатии на кнопку «Ок» пользователю будет предложено выбрать интервал срабатывания стратегии, что можно увидеть на рисунке 45.

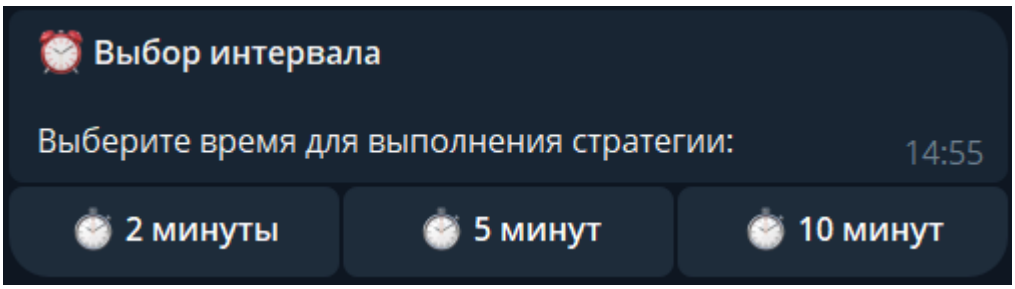

Рисунок 45 - Выбор интервала срабатывания стратегии

Далее пользователь выбирает выбор режима торговли, соответствующие команды представлены на рисунке 46.

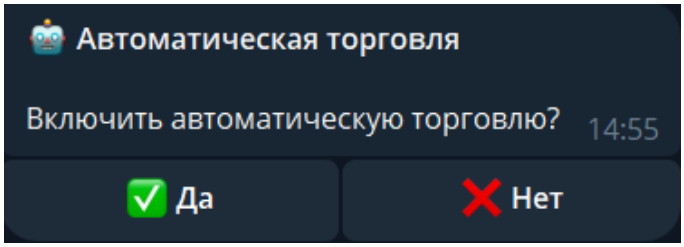

Рисунок 46 - Выбор режима торговли

После выбора режима торговли пользователь должен будет ввести количество бумаг, которые торговый робот будет продавать или покупать за один раз. Данный запрос системы представлен на рисунке 47.

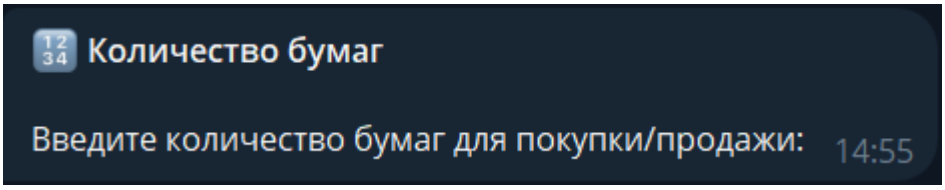

Рисунок 47 - Запрос системы на количество ценных бумаг

Далее пользователь должен выбрать логический оператор срабатывания стратегии. Это означает, что стратегия может сработать либо при уведомлениях на покупку или продажу одновременно всех сигналов, либо же

любого сигнала из выбранных. Окно выбора логического оператора представлено на рисунке 48.

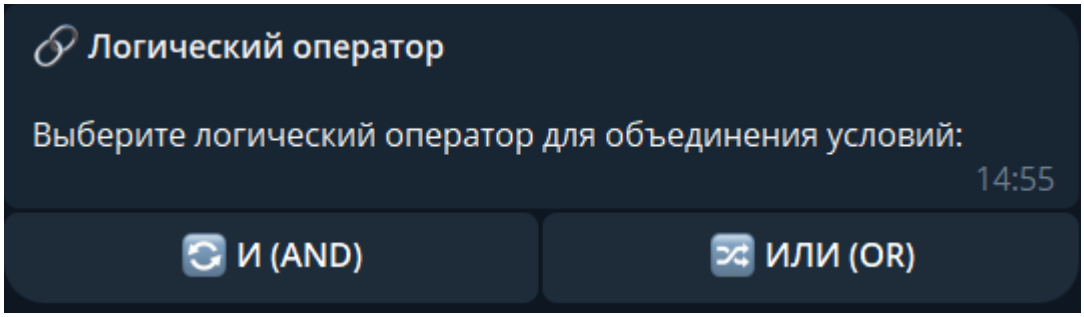

Рисунок 48 - Выбор логического оператора срабатывания

После выбора оператора срабатывания стратегия обновляется и запускается, срабатывая через выбранный временной интервал.

При успешном срабатывании стратегии в терминале отобразится соответствующая запись, а пользователю отправится соответствующее уведомление, пример которого можно увидеть на рисунке 49.

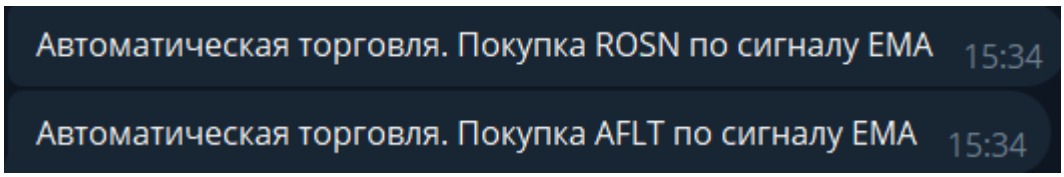

Рисунок 49 - Уведомления об успешной покупке ценных бумаг

Чтобы убедиться в корректности операции, можно зайти в приложение брокера и посмотреть последние выполненные операции, которые представлены на рисунке 50.

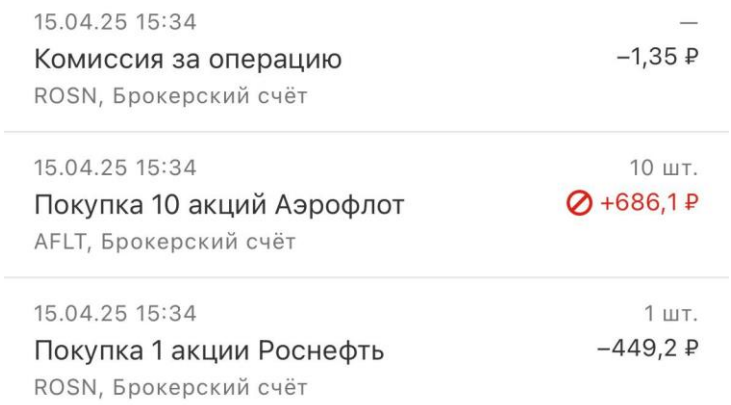

Рисунок 50 - Выполненные операции по покупке ценных бумаг

При выборе команды «Отключить стратегию» все настройки и планировщик стратегии сбрасываются, что можно увидеть на рисунке 51.

Рисунок 51 - Сброшенные настройки стратегии

При выборе команды «Выбор счета» пользователю будет дана возможность выбора из боевого счета и песочницы. При выборе песочницы стратегия пользователя останется такой же, но торги будут вестись уже на отдельном счете с ненастоящей валютой. Если же пользователь выберет боевой счет, торговля будет осуществляться на его основном счете. Окно выбора счета можно увидеть на рисунке 52.

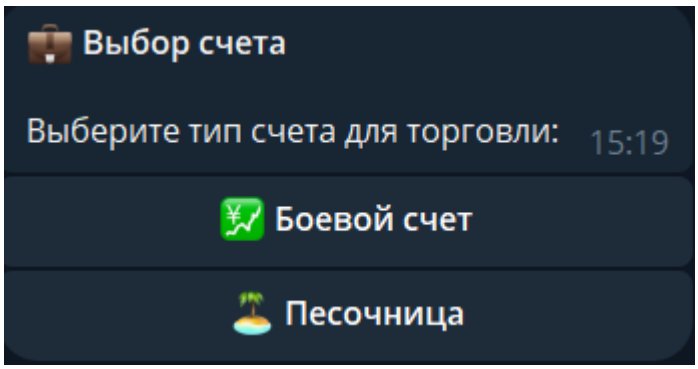

Рисунок 52 - Окно выбора счета

При выборе команды «Информация о песочнице» пользователь получит окно взаимодействия с песочницей, которое представлено на рисунке 53.

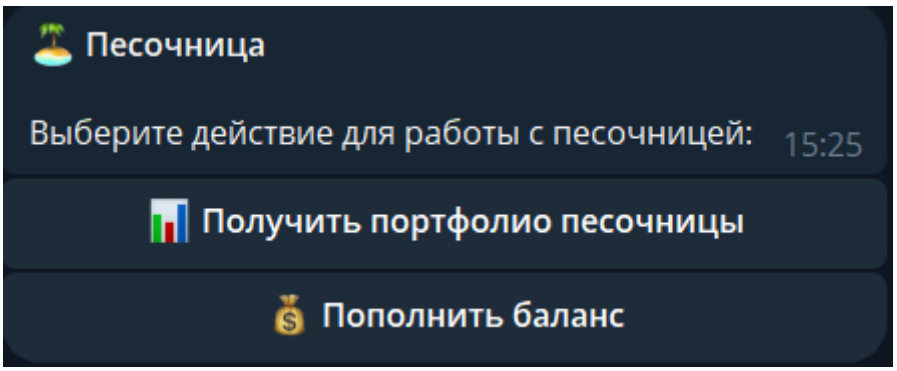

Рисунок 53 - Окно взаимодействия с песочницей

При выборе команды «Пополнить баланс» пользователь сможет ввести значение в рублях, после чего баланс песочницы будет увеличен на это самое значение. Если же пользователь выберет команду «Получить портфолио песочницы», то получит полный отчет по активам в песочнице. Пример такого отчета можно увидеть на рисунке 54.

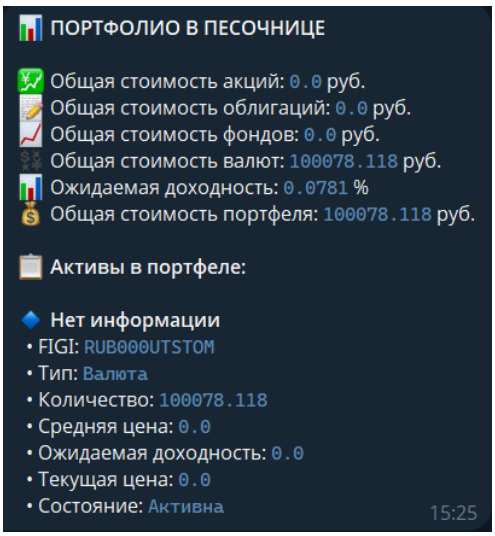

Рисунок 54 - Пример портфолио песочницы

9.7. Модуль дивидендов

Данный модуль представляет из себя функциональность одной команды – получения информации о предстоящих дивидендах по выбранным пользователем инструментам. При выборе команды «Дивиденды» пользователю будет предложено выбрать период, за который он хочет посмотреть наличие дивидендов по ценным бумагам. Соответствующее окно для выбора периода представлено на рисунке 55.

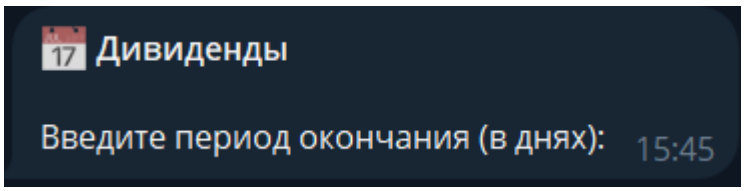

Рисунок 55 - Выбор периода для получения информации по дивидендам

Период получения уведомления по дивидендам в данном случае валидируется и не может быть меньше 1 дня и больше 365 дней. После ввода периода пользователю придет уведомление о дивидендах по каждой ценной бумаге из списка его добавленных инструментов за выбранный им период времени. Пример такого уведомления можно увидеть на рисунке 56.

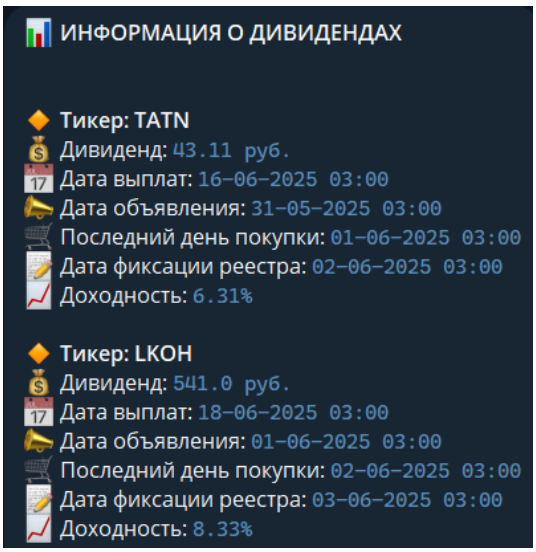

Рисунок 56 - Уведомление о дивидендах

9.8. Модуль долгосрочных и среднесрочных сигналов

Модуль долгосрочных и среднесрочных сигналов предлагает пользователю визуальное представление по конкретным сигналам. При выборе команды по долгосрочным и среднесрочным сигналам пользователю присылается список функциональных кнопок с перечислением сигналов, представленный на рисунке 57.

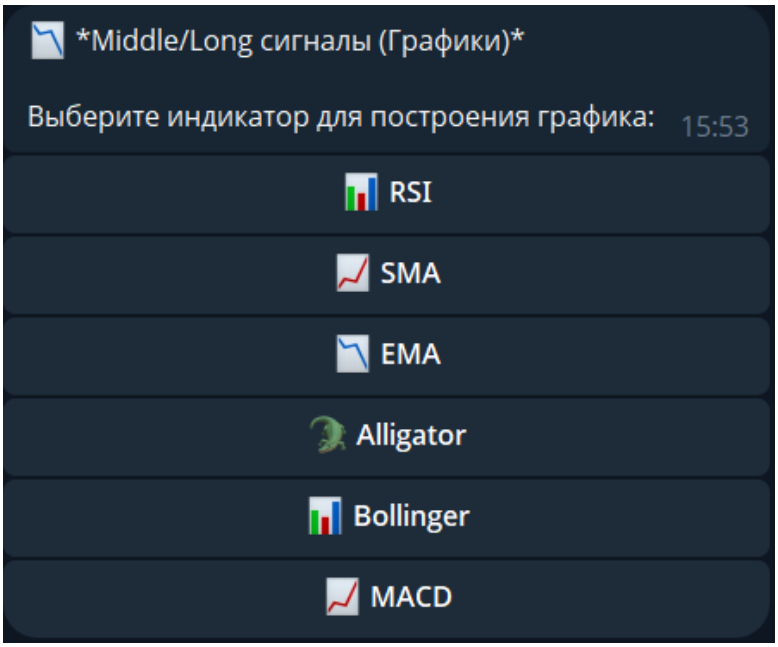

Рисунок 57 - Список доступных сигналов

Если какой-то сигнал не был настроен, то при его выборе пользователю отправится соответствующее уведомление, представленное на рисунке 58.

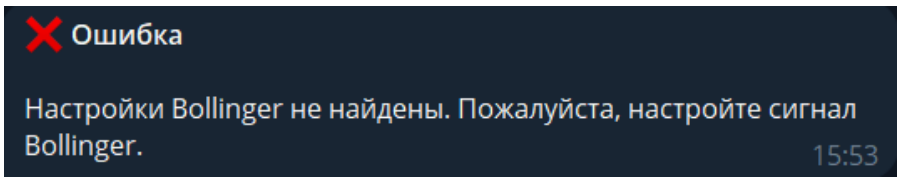

Рисунок 58 - Уведомление о настройке сигнала

В качестве примера будет рассмотрен сигнал SMA. После выбора соответствующего сигнала пользователю будет предложен выбор из временных периодов, который представлен на рисунке 59.

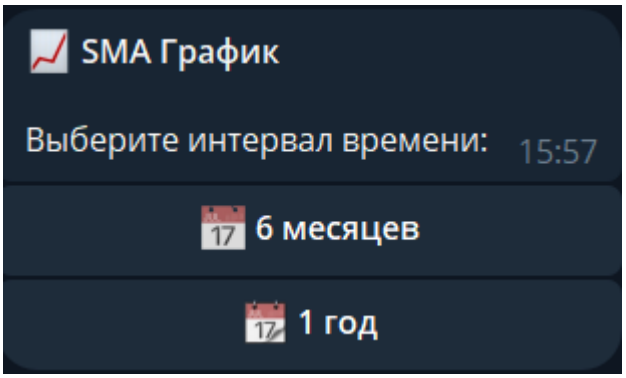

Рисунок 59 - Выбор временного периода для сигнала

После выбора периода пользователю будет представлен список функциональных кнопок, состоящий из перечисления его ценных бумаг. Данный список представлен на рисунке 60.

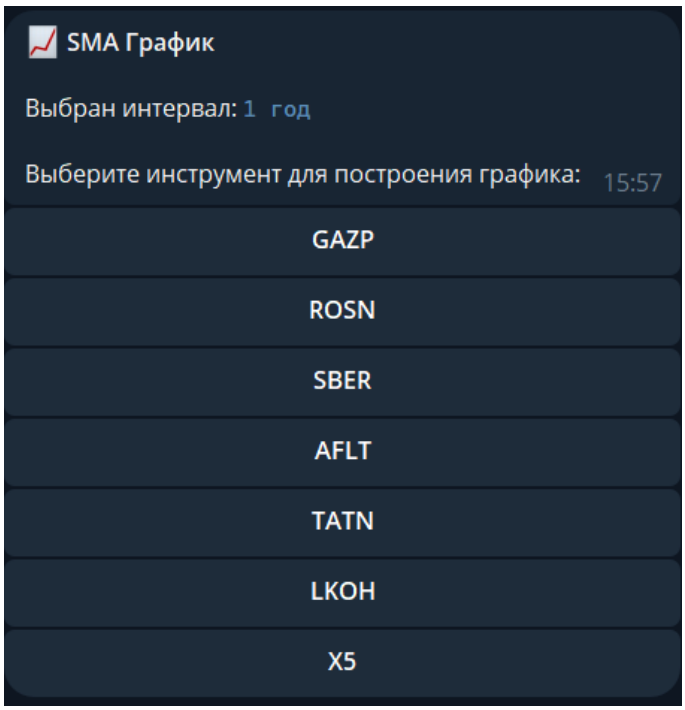

Рисунок 60 - Список ценных бумаг пользователя

После выбора пользователем нужной ему ценной бумаги, будет построен график по выбранному сигналу, отражающий временной промежуток и все параметры сигнала, а также ценовые свечи инструмента. Пример такого графика представлен на рисунке 61. Программный код функции для построения графика представлен в приложении 9.

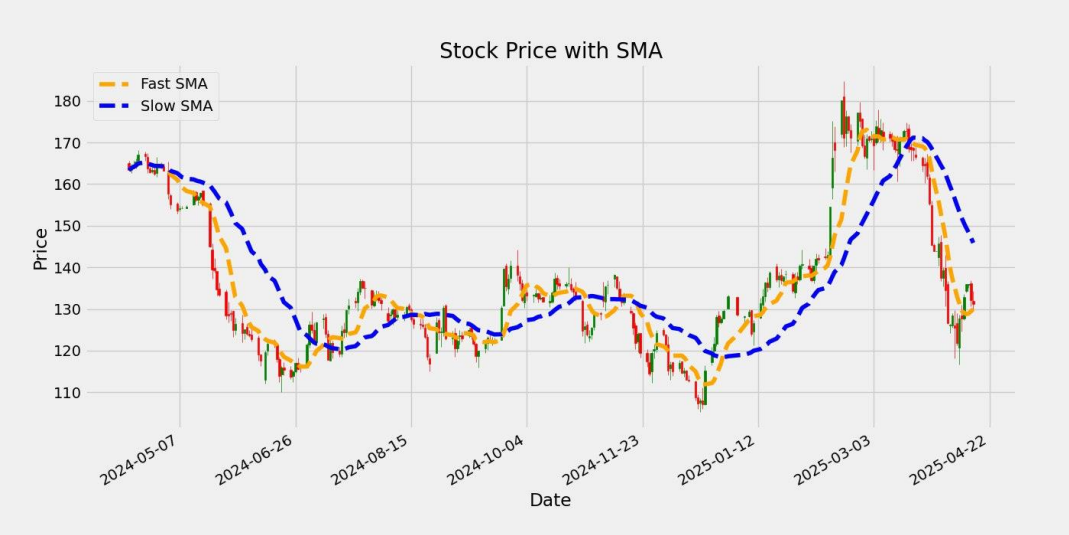

Рисунок 61 - График сигнала SMA

9.9. Модуль базы знаний

Модуль базы знаний предоставляет пользователю набор информации по всем модулям системы. При выборе команды «База знаний» пользователь получает уведомление с функциональными кнопками, которое представлено на рисунке 62.

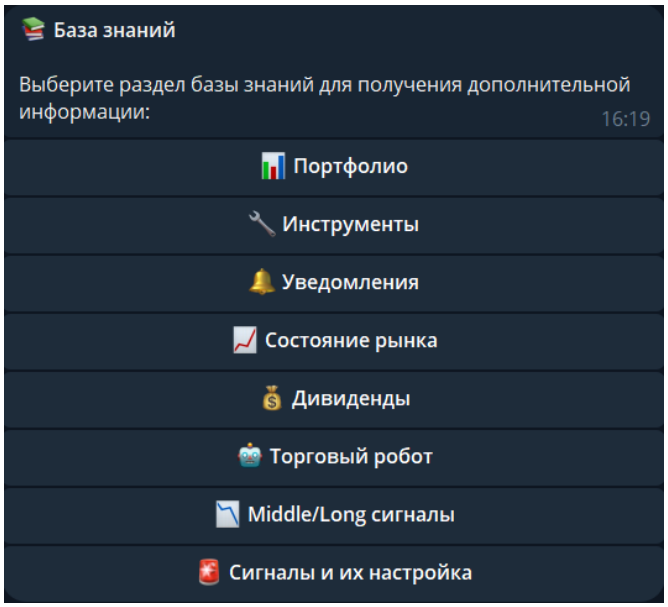

Рисунок 62 - База знаний

При выборе необходимой пользователю команды он получит информацию по интересующему его модулю. Для примера будет рассмотрена команда «Инструменты». При выборе данной команды пользователь получит уведомление, представленное на рисунке 63.

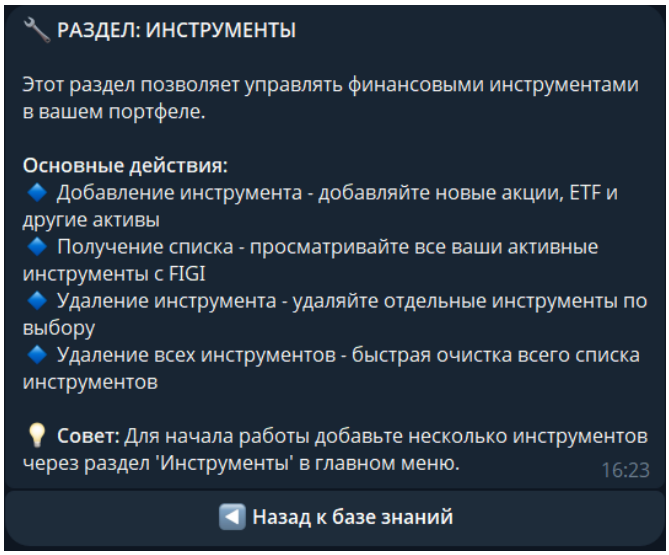

Рисунок 63 - Раздел инструментов

В данном разделе пользователь может найти общую информацию по соответствующему модулю, обзор выполняемых модулем команд, а также некоторые советы по работе данного модуля внутри системы.

9.10. Модуль статистики

Данный модулю предоставляет пользователю статистику покупок и продаж, совершенных через торгового робота. При выборе команды «Статистика» пользователю будет предложено либо выбрать интервал получения статистики, либо же получить общую статистику за все время. Окно выбора типа статистики представлено на рисунке 64.

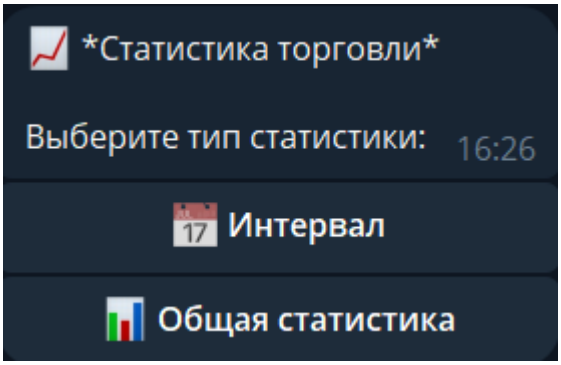

Рисунок 64 - Окно выбора типа статистики

После выбора получения интервала статистики пользователю будет отправлено наглядное представление покупок и продаж в виде различных графиков, которые будут представлены в ответе в различных видах. Для построения графиков используется библиотека matplotlib [8].

На первом типе графиков отражена статистика покупок или продаж по тикерам. Пример такого графика можно увидеть на рисунке 65.

Рисунок 65 - График количества покупок по тикерам

Второй тип графиков отражает количество покупок или продаж по сигналам. Пример такого графика можно увидеть на рисунке 66.

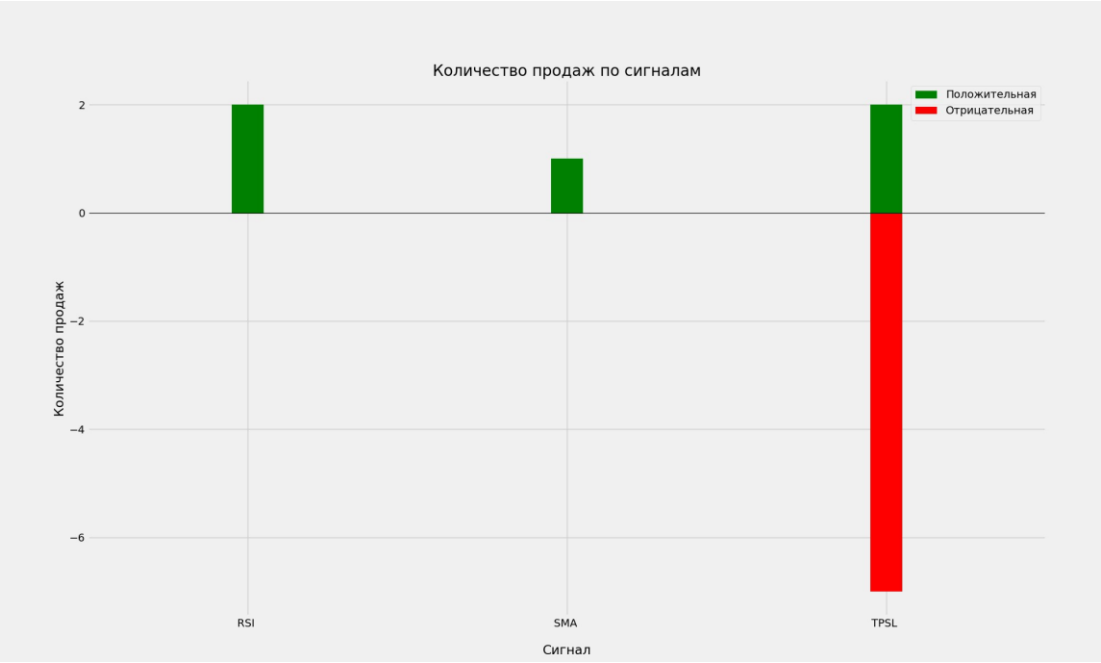

Рисунок 66 - График количества продаж по сигналам

Следующий тип графиков отражает количество покупок или продаж по дням. Пример такого графика представлен на рисунке 67.

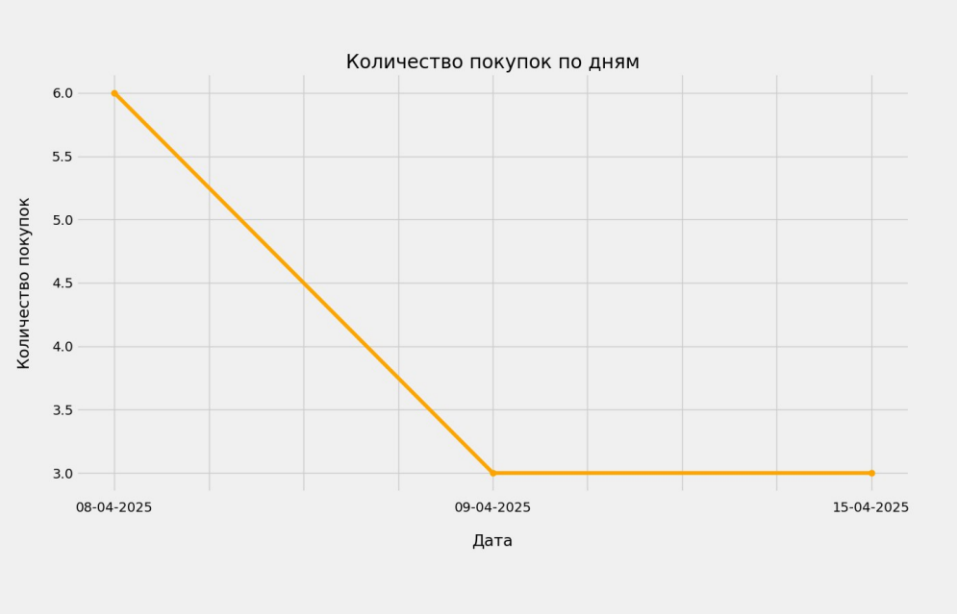

Рисунок 67 - График количества покупок по дням

Последний график – это график соотношения положительной и отрицательной маржи. Пример такого графика представлен на рисунке 68.

Рисунок 68 - График соотношения маржи

Программный код функции для визуального представления статистики представлен в приложении 10.

10. Анализ разработанной системы

10.1. Анализ пользовательского взаимодействия с системой

Разработанная система ориентирована в первую очередь на конечного пользователя. Важным аспектом её оценки является анализ с точки зрения удобства, понятности и доступности взаимодействия со стороны потребителя. Как и любое программное обеспечение, данная система обладает как положительными, так и отрицательными сторонами, влияющими на пользовательский опыт.

Одним из ключевых преимуществ системы является возможность гибкой настройки параметров, что можно увидеть на рисунке 69.

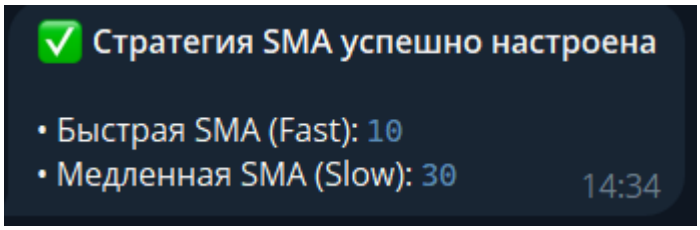

Рисунок 69 - Конечный этап настройки сигнала SMA

Такая настройка позволяет адаптировать поведение системы под индивидуальные предпочтения пользователя, что очень удобно в сфере автоматизированной торговли, где каждая стратегия может требовать уникального набора параметров. Пользователю доступен полный контроль над конфигурацией сигналов и условий исполнения торговых алгоритмов, что в совокупности создаёт пространство для экспериментов и поиска оптимальных решений, не ограничиваясь жёсткими рамками заранее заданной логики.

Из преимуществ системы также можно выделить наличие в ней «песочницы» — специального режима работы системы, предназначенного для тестирования различных сценариев без риска совершения реальных операций на бирже. Благодаря этой функции пользователь может проверить корректность работы настроенных им сигналов и провести полноценный анализ стратегий, оценить их эффективность и внести необходимые

корректировки до перехода к реальной торговле. Варианты выбора счетов представлены на рисунке 70.

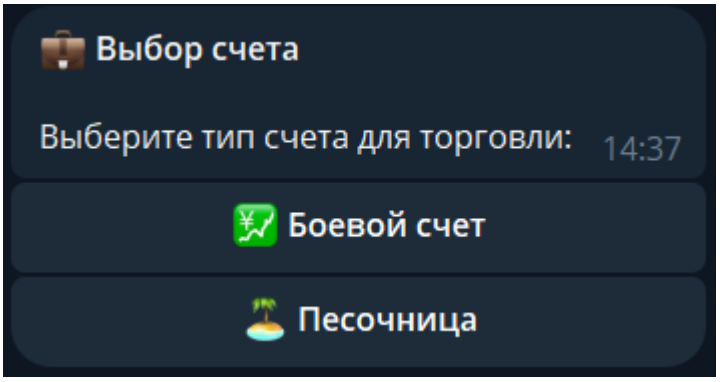

Рисунок 70 – Выбор пользовательского счета

Интерфейс системы построен таким образом, чтобы все действия пользователя были сосредоточены в одном месте. Все пользовательские команды находятся в одной клавиатуре, разделенной на функциональные модули приложения. При взаимодействии с системой большого нагромождения сообщений не будет, так как работа с одним конкретным функциональным модулем происходит в рамках одного сообщения, которое меняет свой внешний вид и наполнение в зависимости от действий пользователя. Пример последовательного изменения сообщения можно увидеть на рисунке 71 и на рисунке 72.

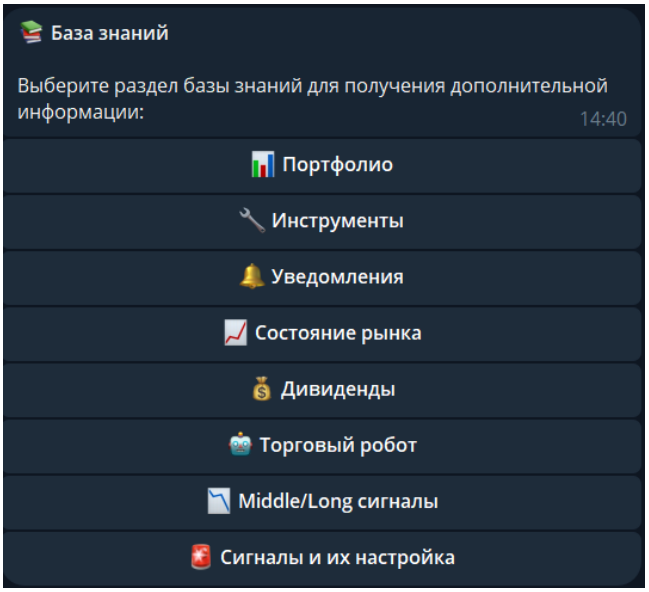

Рисунок 71 - Начальное состояние текущего сообщения

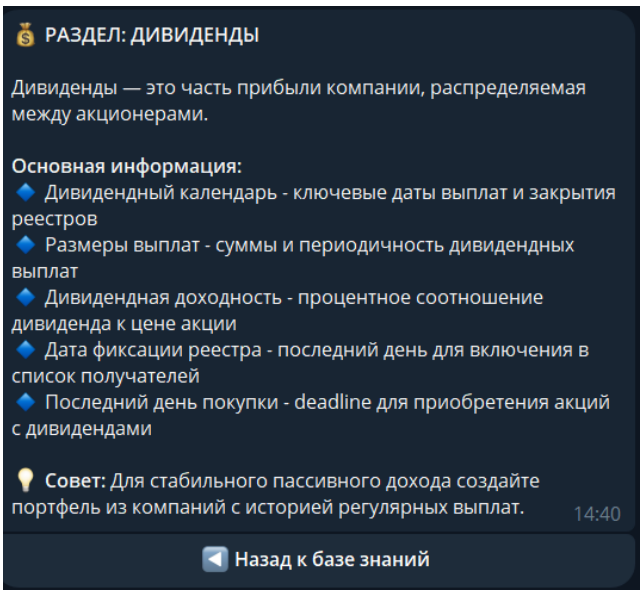

Рисунок 72 - Конечное состояние текущего сообщения

Не менее важной частью пользовательского опыта является информирование. На каждую операцию у системы есть определенный ответ, который будет отправляться пользователю. Если торговый робот совершил какую-либо операцию на бирже, то пользователь получит об этом уведомление, что можно увидеть на рисунке 73. Если произошла какая-либо ошибка, пользователь будет об этом проинформирован.

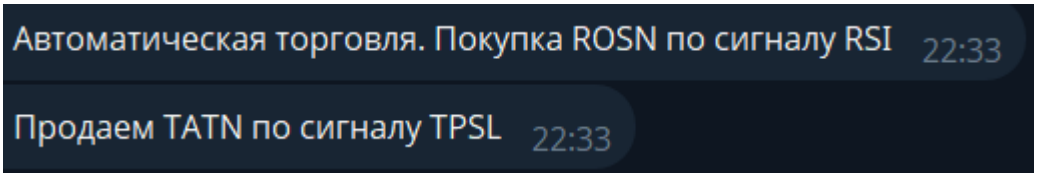

Рисунок 73 - Уведомления об операциях на бирже

Наличие в системе базы данных обеспечивает сохранность всех пользовательских настроек и результатов торговли. Это делает возможным восстановление конфигурации при повторных запусках, а также дает пользователю возможности для построения анализа торговли, выявляя закономерности и оценивая эффективность принятых решений.

Несмотря на ряд безусловных достоинств, система также имеет и некоторые недостатки. Одной из самых ощутимых проблем является зависимость от стабильного интернет-соединения. При потере связи система может перестать выполнять запросы, а при запуске в локальном режиме без использования Docker-контейнера не предусмотрено автоматическое

восстановление работы, что может стать критичным фактором в моменты активной торговли.

Стабильность работы интерфейса напрямую зависит от Telegram API, через который осуществляется взаимодействие. В отдельных случаях задержки между действиями пользователя и ответом системы могут быть заметны, что способно вызывать ощущение «подвисания» системы и снижать уровень удовлетворённости от использования. В дальнейшем планируется перенос системы с библиотеки telebot на более продвинутую библиотеку aiogram, поддерживающую асинхронное взаимодействие [9].

Как отдельный недостаток можно выделить ограниченность интерфейса. В виду ограничений интерфейса приложения telegram, не является возможным внедрить в приложение различные всплывающие подсказки, удобное навигационное меню или любые другие различные элементы интерфейса, облегчающие работу пользователей.

Дополнительной сложностью можно считать информационную перегруженность отдельных модулей. Например, при получении информации о портфеле пользователю выводятся все доступные данные, включая такие параметры, как средняя цена покупки или статус актива, что можно увидеть на рисунке 74. Для опытных пользователей такие сведения могут быть полезны, при этом начинающие трейдеры могут испытывать затруднения при интерпретации большого объёма информации.

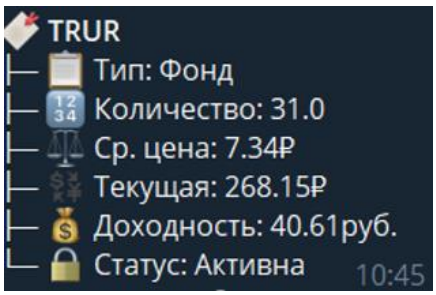

Рисунок 74 - Пример позиции в портфолио

10.2. Анализ архитектуры разработанной системы

Архитектура разработанной системы представляет собой важную составляющую, которая определяет как внутреннюю организацию

компонентов системы, так и ее дальнейшие перспективы масштабирования, сопровождения и адаптации под изменяющиеся требования.

В первую очередь необходимо сказать, что система построена на принципах модульности [10]. Структура разработанной системы представлена на рисунке 75.

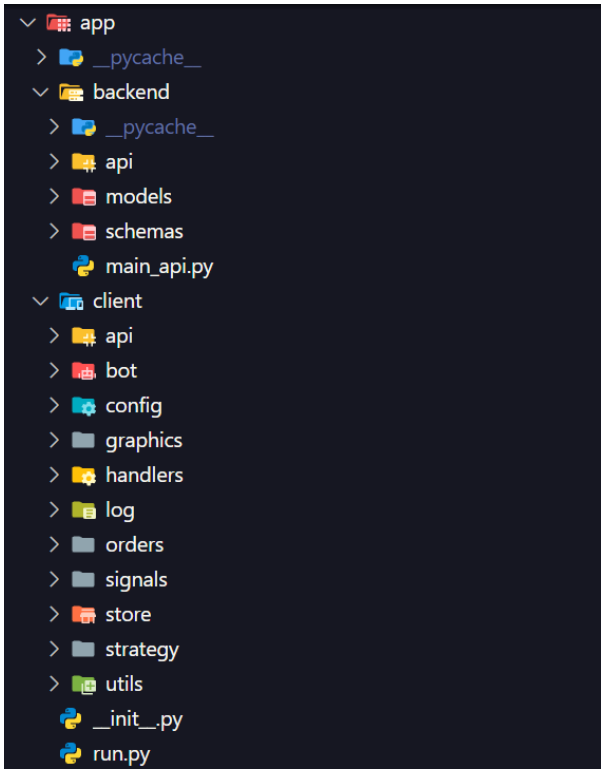

Рисунок 75 - Структура приложения

Каждый функциональный компонент приложения выделен в отдельную папку, а логика внутри файлов структурирована так, чтобы обеспечивать изоляцию конкретных задач. Каждый модуль системы отвечает строго за свою область ответственности, что делает такой подход подходящим для лучшего пониманию структуры кода, упрощает масштабирование в будущем, а также облегчает тестирование и внедрение новых функций.

Несмотря на очевидные преимущества модульной структуры, данный подход имеет свои недостатки. Одним из них является избыточное количество модулей и хранилищ для различных компонентов, что приводит к увеличению объема кода и усложнению навигации в проекте. Таким подходом с разделением ответственности под каждый модуль возможно перегрузить систему избыточной детализацией, особенно если структура недостаточно

стандартизирована или дублирует однотипные подходы в разных модулях. Стоит отметить, что модульность в текущей реализации не исключает наличие монолитности [11] на уровне всей архитектуры. Несмотря на логическую изоляцию компонентов и разделение ответственности, система по-прежнему функционирует как единое целое, где все модули запускаются и взаимодействуют внутри одного процесса.

Подобная архитектура в долгосрочной перспективе может стать ограничением. С внедрением многопользовательской архитектуры и ростом числа функциональных возможностей возникает необходимость в более гибком и масштабируемом подходе. Наиболее подходящим в таком контексте представляется переход к микросервисной архитектуре [12]. В рамках микросервисного подхода каждый функциональный модуль может быть преобразован в отдельный сервис, взаимодействующий с другими через API, что позволит упростить развертывание обновлений и ускорить масштабирование отдельных компонентов в зависимости от нагрузки.

Отдельной позицией можно рассмотреть реализацию пользовательского интерфейса. Основной средой взаимодействия пользователя с системой выступает Telegram, что делает приложение доступным и удобным в использовании на любом устройстве. Однако Telegram-интерфейс обладает как своими преимуществами, так и ограничениями. Он удобен для быстрой работы и обмена сообщениями, но ограничен в визуальных возможностях, таких как всплывающие окна, интерактивные графические элементы и сложные навигационные компоненты. Сознательно учитывая эти ограничения, при проектировании архитектуры было принято решение разделить систему на две ключевые части: backend и интерфейсную. Backend отвечает за выполнение бизнес-логики, обработку запросов, хранение данных и генерацию ответов. Интерфейсная часть, основанная на Telegram API, реализует пользовательские сценарии, принимая команды и отображая ответы в виде сообщений, кнопок и интерактивных элементов Telegram-бота.

Такой подход обеспечивает гибкость в дальнейшем развитии системы. Пользователь, обладающий необходимыми навыками, может заменить Telegram-интерфейс на любой другой — например, реализованный на базе фреймворков React, Vue или даже в виде мобильного приложения [13]. Весь набор бизнес-операций уже вынесен в виде API, и при необходимости может быть повторно использован или расширен.

Если обобщить результаты анализа по ключевым критериям, то преимущества и недостатки разработанной системы можно наглядно представить в виде следующей таблицы (таблица 1):

Таблица 1 - Сравнительный анализ разработанной системы

Критерий	Преимущества	Недостатки
Интерфейс пользователя	Удобный, интуитивно понятный Telegram-интерфейс, использующий изменяемые сообщения	Отсутствие кастомизации интерфейса, внедрения подсказок и удобных пользовательских компонентов, нестабильность в некоторых случаях
Гибкость настройки	Расширенные возможности конфигурации стратегий и сигналов. Пользователь может адаптировать систему под свои нужды	Может быть нужно не каждому пользователю, нет уже готовых настроек
Открытость	Открытый исходный код позволяет пользователям вносить изменения, модифицировать поведение и анализировать алгоритмы	Отсутствие технической поддержки, как у коммерческих решений. Требует самостоятельного изучения кода при глубокой кастомизации

Продолжение таблицы 1

Критерий	Преимущества	Недостатки
Функциональность	Содержит в себе все необходимые трейдеру функции для работы с личным портфелем, избранными ценными бумагами и торговым роботом	Для некоторых пользователей может быть избыточной, а для продвинутых пользователей наоборот, недостаточной
Независимость от разработчиков	Пользователь контролирует всё — нет зависимости от внешней технической поддержки, можно работать автономно	Необходимы базовые технические знания для устранения проблем
Архитектура	Модульная архитектура с разбиением на зоны ответственности	При дальнейшем расширении может возникнуть чрезмерная избыточность. Основа всей системы – монолит, при масштабировании необходимо рассмотреть вариант микросервисной архитектуры

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была реализована интерактивная система, предназначенная для автоматизированного трейдинга и управления финансовыми активами. Система позволяет пользователю в удобной форме получать информацию о состоянии инвестиционного портфеля, следить за изменениями на рынке, настраивать технические сигналы, а также запускать торгового робота, действующего в соответствии с заданной стратегией.

Разработка была выполнена с учётом требований к масштабируемости, модульности и простоте взаимодействия. Также особое внимание было уделено гибкости архитектуры, позволяющей адаптировать систему под индивидуальные сценарии использования путем изменения различных параметров и настроек, что обеспечивает широкие возможности по управлению логикой работы сигналов и стратегий в зависимости от предпочтений конкретного пользователя.

На этапе тестирования и в процессе эксплуатации проверялась устойчивость всех компонентов системы к типовым пользовательским ситуациям, а также стабильность взаимодействия с брокерским API, что подтвердило корректность обработки пользовательских запросов и выполнение торговых стратегий в соответствии с заданными параметрами.

Полученные результаты говорят о том, что созданная система может эффективно использоваться в рамках реального инвестиционного процесса. Она способна существенно снизить трудозатраты на рутинные операции, предоставить пользователю актуальную информацию в удобной форме и повысить качество принимаемых решений за счёт системного подхода к анализу данных и автоматизации торговли.

Таким образом, разработанная система может служить надёжным инструментом и помощником для частных инвесторов, стремящихся автоматизировать торговые процессы и повысить эффективность своей инвестиционной деятельности. Полученные результаты создают основу для

дальнейшего развития проекта и возможного расширения его функциональности в будущем, включая внедрение новых видов сигналов, работу с другими брокерскими платформами, интеграцию с веб-интерфейсами и аналитическими сервисами.

За время выполнения выпускной квалификационной работы были реализованы следующие компетенции (таблица 2):

Таблица 2 - Приобретенные компетенции

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Сравнивал данные, полученные через систему, с данными официального брокера. Оценивал корректность работы функций и отклонения в расчетах
УК-2	Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Определил ключевые задачи разработки системы, выбрал оптимальные методы реализации с учетом ограничений и доступных ресурсов
УК-3	Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Взаимодействовал с научным руководителем, обсуждал промежуточные результаты и корректировал реализацию системы на основе полученной обратной связи

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-4	Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	Осуществлял переписку, поиск информации и консультации по техническим вопросам на русском и английском языках, оформлял документацию по проекту в письменной форме
УК-5	Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	При разработке интерфейса учитывал универсальность и нейтральность формулировок, чтобы система подходила пользователям с разным социальным и культурным фоном
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Определил ключевые задачи разработки системы, выбрал оптимальные методы реализации с учетом ограничений и доступных ресурсов. Организовывал процесс разработки с четким планированием задач и соблюдением сроков, улучшая свои навыки управления временем

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-7	Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	Соблюдал баланс между учебной и физической активностью, что позволяло поддерживать высокую работоспособность на протяжении всего периода разработки
УК-8	Способен создавать и поддерживать в повседневной жизни и в профессиональной деятельности безопасные условия жизнедеятельности для сохранения природной среды, обеспечения устойчивого развития общества, в том числе при угрозе и возникновении чрезвычайных ситуаций и военных конфликтов	При разработке системы использовал энергоэффективные вычислительные ресурсы и средства защиты данных, минимизируя риски и нагрузку на инфраструктуру
УК-9	Способен принимать обоснованные экономические решения в различных областях жизнедеятельности	Принял решение в пользу использования бесплатных и открытых технологий при разработке системы, чтобы минимизировать затраты без ущерба для качества

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-10	Способен формировать нетерпимое отношение к проявлениям экстремизма, терроризма, коррупционному поведению, противодействовать им в профессиональной деятельности	Соблюдал этические нормы и нормы академической честности при выполнении всех этапов проектирования и разработки
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	Использовал знания в области теории систем для анализа архитектуры и обработки данных в программном комплексе
ОПК-2	Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности	Применил инструменты анализа программного обеспечения и визуализации данных для оценки функциональности системы

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ОПК-3	Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям	Разработал тестовые сценарии и провёл моделирование пользовательского поведения для оценки отклика и устойчивости системы
ОПК-4	Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла	Сформировал аналитическую часть отчётной документации по результатам анализа, дал рекомендации по улучшению архитектуры и повышению устойчивости к ошибкам

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ОПК-5	Способен инсталлировать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности	Настроил окружение для развёртывания приложения с использованием Docker,
ОПК-6	Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности	Использовал современные технологии для построения и отладки всех компонентов системы
ПК-1	Проверка работоспособности и рефакторинг кода программного обеспечения	Проводил ручное тестирование всех модулей системы, выявлял ошибки отображения, некорректные состояния и обрабатывал пограничные случаи
ПК-2	Интеграция программных модулей и компонент и верификация выпусков программного продукта	Проверял согласованную работу всех компонентов системы

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ПК-3	Разработка требований и проектирование программного обеспечения	На основе результатов тестирования сформулировал предложения по улучшению логики работы команд и взаимодействия с пользователем
ПК-4	Оценка и выбор варианта архитектуры программного средства	Запускал систему через основной исполняемый файл, анализировал взаимодействие компонентов, оценивал удобство сопровождения и тестирования выбранной архитектурной схемы
ПК-5	Разработка тестовых случаев, проведение тестирования и исследование результатов	Разработал сценарии тестирования, проверил корректность работы команд и реакцию системы на типовые и граничные входные данные
ПК-6	Обеспечение и оптимизация функционирования баз данных	Оптимизировал структуру базы данных и сформулировал SQL-запросы с учётом быстродействия при увеличении данных

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ПК-7	Обеспечение информационной безопасности на уровне базы данных	Реализовал меры защиты данных пользователей, исключив доступ к чувствительной информации из внешних источников, за счёт использования токенов и проверок на уровне API
ПК-8	Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы	Разработал систему для автоматизации задач, интегрируя внешние сервисы и создавая алгоритмы для обработки данных
ПК-9	Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности	Сформулировал требования и написал технические задания для разработки и настройки системы с учетом всех функциональных потребностей
ПК-10	Способен к коммуникации, восприятию информации, умению логически верно, аргументировано и ясно строить устную и письменную речь для решения профессиональных задач	Подготовил отчётную документацию по системе, изложил технические детали в ясной и логичной форме, объяснял логику работы компонентов во время консультаций

Продолжение таблицы 2

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ПК-11	Способен использовать действующее законодательство и другие правовые документы в своей деятельности, демонстрировать готовность и стремление к совершенствованию и развитию общества на принципах гуманизма, свободы и демократии	Изучил условия лицензирования и правомерность использования API-брокера, учитывал требования безопасности при работе с пользовательскими данными

СПИСОК ЛИТЕРАТУРЫ

1) SQLite Documentation [Электронный ресурс]. – 2025. — URL: https://www.sqlite.org/docs.html (дата обращения 10.03.2025).

2) SQLAlchemy – The Database Toolkit for Python [Электронный ресурс]. – 2023. — URL: https://www.sqlalchemy.org/ (дата обращения 31.03.2025).

3) Docker: как развернуть фуллстек-приложение [Электронный ресурс]. – 2019. — URL: https://habr.com/ru/articles/448094/ (дата обращения 05.04.2025).

4) FastAPI Documentation [Электронный ресурс]. – 2024. — URL: https://fastapi.tiangolo.com/ (дата обращения 24.03.2025).

5) T-Bank Invest API [Электронный ресурс]. – 2023. — URL: https://tinkoff.github.io/investAPI/ (дата обращения 12.03.2025).

6) Python Logging Module Documentation [Электронный ресурс]. – 2025. — URL: https://docs.python.org/3/library/logging.html (дата обращения 14.04.2025).

7) Investopedia. Technical Indicators: SMA, RSI, MACD [Электронный ресурс]. – 2023. — URL: https://www.sqlalchemy.org/ (дата обращения 20.04.2025).

8) Matplotlib documentation [Электронный ресурс]. – 2024. — URL: https://matplotlib.org/stable/index.html (дата обращения 22.04.2025).

9) Aiogram documentation [Электронный ресурс]. – 2025. — URL: https://docs.aiogram.dev/ (дата обращения 10.05.2025).

10) Правильный подход к модульной архитектуре [Электронный ресурс]. – 2024. — URL: https://habr.com/ru/articles/799169/ (дата обращения 15.05.2025).

11) Хорошие монолиты. Простая архитектура лучше всего [Электронный ресурс]. – 2024. — URL: https://habr.com/ru/companies/ /articles/676780/ (дата обращения 19.05.2025).

12) Микросервисная архитектура [Электронный ресурс]. – 2024. — URL: https://www.atlassian.com/ru/microservices/microservices-architecture (дата обращения 22.05.2025).

13) Vue 3 documentation [Электронный ресурс]. – 2025. — URL: https://vuejs.org/guide/introduction (дата обращения 24.05.2025).

ПРИЛОЖЕНИЯ

Приложение 1. Код создания торгового поручения на покупку

```
        avaliable_lots = calc_avaliable_lots(token, figi,
client, sandbox_method)
                    # Проверка на наличие позиции в портфеле
                    if (avaliable_lots > 0):
                        print(f"Позиция {figi} уже в портфеле,
ждем сигнала к продаже...")
                        return

                    # best or fast
                    price_sell, price_buy =
get_current_price(figi, client, 'fast')
                    # Проверка баланса
                    if check_enough_currency(token, figi,
client, price_buy, quantity, sandbox_method):
                        # Генерация id
                        order_id = str(uuid.uuid4())

                        r = sb.post_sandbox_order(
                            figi=figi,
                            quantity=quantity,
                            price=price_buy,
                            account_id=account_id,
                            order_id=order_id,

direction=OrderDirection.ORDER_DIRECTION_BUY,

order_type=OrderType.ORDER_TYPE_LIMIT,
                        )

                        # Создаем новый ордер через API клиент
                        api_client.add_order({
                            "order_id": order_id,
                            "ticker": ticker,
                            "signal": signal,
                            "bm_value": cast_money(price_buy),
                            "operation_type": operation
                        })

                        logger.info(r)
                        logger.info(f"Создаем заявку на покупку
по цене {cast_money(price_buy)}")

            except RequestError as e:
                logger.error(str(e))
```

Приложение 2. Программный код маршрута для добавления нового инструмента

```
@router.post("/", response_model=InstrumentResponse)
def create_instrument(instrument: InstrumentCreate, db: Session
= Depends(get_db)):
    """
    Создать новый инструмент.
    """
    existing_instrument =
db.query(Instrument).filter(Instrument.ticker ==
instrument.ticker).first()
    if existing_instrument:
        raise HTTPException(status_code=400, detail="Instrument
with this ticker already exists")

    db_instrument = Instrument(**instrument.dict())
    db.add(db_instrument)
    db.commit()
    db.refresh(db_instrument)
    return db_instrument
```

Приложение 3. Программный код API-запроса для добавления нового инструмента

```
def add_instrument(self, ticker: str, figi: str) -> Dict[str,
Any]:
        """
        Добавляет новый инструмент.

        Args:
            ticker: Тикер инструмента
            figi: FIGI инструмента

        Returns:
            Dict[str, Any]: Добавленный инструмент

        Raises:
            requests.exceptions.HTTPError: Если произошла ошибка
HTTP при добавлении инструмента
        """
        try:
            data = {"ticker": ticker, "figi": figi}
            return self._post("instruments/", data)
        except requests.exceptions.HTTPError as e:
            print(f"HTTP Error: {e}")
            raise
```

Приложение 4. Функция вычисления изменения цены

```
def get_price_change_in_current_interval(figi, start_time,
end_time, candle_interval):

    """
    Вычисляет изменение цены для заданного интервала времени.

    :param figi: Строка, идентификатор финансового инструмента.
    :param start_time: Начальное время интервала.
    :param end_time: Конечное время интервала.
    :param candle_interval: Интервал свечи для получения данных.

    """

    try:

        data = get_historic_candles(figi, start_time, end_time,
candle_interval)

        df = create_df(data.candles)

        # Проверяем, есть ли данные в DataFrame
        if df.empty:
            print("Нет данных за указанный период")
            return None

        # Получаем цену открытия и цену закрытия
        open_price = df['open'].iloc[0]
        close_price = df['close'].iloc[-1]

        max_price = df['high'].max()
        min_price = df['low'].min()

        # Рассчитываем изменение цены
        price_change = close_price - open_price

        # Рассчитываем процентное изменение цены
        price_change_percent = (price_change / open_price) * 100

        # Выводим результат
        logger.info(f"Изменение цены за период:
{price_change:.2f} ({price_change_percent:.2f}%)")
        logger.info(f"Максимальная цена: {max_price:.2f}\n
Минимальная цена: {min_price:.2f}")

        # Возвращаем результат
        return price_change, price_change_percent, max_price,
min_price, close_price

    except RequestError as e:
        logger.error(f"Ошибка запроса: {e}")
        return None
```

Приложение 5. Программный код реализации сигнала MACD

```
def calculate_macd_signal(data, fast_length, slow_length,
signal_length, profit):
    """
    Вычисление MACD и генерация торгового сигнала.
    :param data: Данные о свечах.
    :param fast_length: Период для быстрой EMA.
    :param slow_length: Период для медленной EMA.
    :param signal_length: Период для сигнальной линии.
    :param profit: Текущая прибыль.
    :return: Торговый сигнал ('buy', 'sell', 'hold').
    """
    # Создаем DataFrame из данных о свечах
    candles = data.candles
    df = create_df(candles)
    # Преобразуем цены закрытия в Series
    close_prices = pd.Series(df['close'].values)
    # Рассчитываем MACD
    macd_values = ta.trend.MACD(close_prices,
window_slow=slow_length, window_fast=fast_length,
window_sign=signal_length)

    # Получаем линии MACD и сигнальную линию
    macd_line = macd_values.macd().values
    signal_line = ta.trend.ema_indicator(pd.Series(macd_line),
window=signal_length, fillna=True).to_numpy()

    store.signal_line = signal_line
    store.macd_line = macd_line

    # Проверяем, что длины массивов совпадают
    if len(macd_line) != len(signal_line):
        raise ValueError("Длина линии MACD и сигнальной линии не
совпадают.")

    # Получаем последние значения
    current_macd = macd_line[-1]
    current_signal = signal_line[-1]

    # Генерация торгового сигнала
    if current_macd > current_signal:  # Условие для покупки
        return 'buy'
    elif current_macd < current_signal and profit > 0:  #
Условие для продажи
        return 'sell'
    else:
        return 'hold'
```

Приложение 6. Программный код реализации сигнала Alligator

```
def calculate_alligator_strategy(data, jaw_period, jaw_shift,
teeth_period, teeth_shift, lips_period, lips_shift, profit):
    """
    Функция для расчета стратегии Аллигатора (Alligator) Bill
Williams.
    Она использует три скользящие средние: челюсти, зубы и губы,
    которые демонстрируют возможное местонахождение цены при
отсутствии фундаментальных новостей.

    :param data: исторические данные ( HistoricCandle ) для
расчета стратегии
    :param jaw_period: период для расчета Челюстей
    :param jaw_shift: сдвиг Челюстей
    :param teeth_period: период для расчета Зубов
    :param teeth_shift: сдвиг Зубов
    :param lips_period: период для расчета Губ
    :param lips_shift: сдвиг Губ
    :param profit: прибыль для определения момента для продажи

    :return: 'buy', 'sell' или 'hold', в зависимости от
стратегии
    """
    candles = data.candles
    df = create_df(candles)

    # Рассчитываем среднюю цену по каждому бару: (High + Low) /
2
    avg_prices = (df['high'].values + df['low'].values) / 2

    # Рассчитываем скользящие средние для челюстей, зубов и губ
на основе средней цены
    jaw_sma = sma(avg_prices, jaw_period)
    teeth_sma = sma(avg_prices, teeth_period)
    lips_sma = sma(avg_prices, lips_period)

    # Применяем смещения: сдвигаем массивы на заданное
количество баров
    jaw_sma = np.roll(jaw_sma, jaw_shift)
    teeth_sma = np.roll(teeth_sma, teeth_shift)
    lips_sma = np.roll(lips_sma, lips_shift)

    store.jaw_sma = jaw_sma
    store.teeth_sma = teeth_sma
    store.lips_sma = lips_sma

    # Проверяем пересечения с учетом положения линий
    # Покупка при пересечении губ с зубами и зубов с челюстями
снизу вверх
    if crossover(lips_sma, teeth_sma) and crossover(teeth_sma,
jaw_sma):
        return 'buy'
```

```
    # Продажа при пересечении губ с зубами и зубов с челюстями
сверху вниз
    if crossunder(lips_sma, teeth_sma) and crossunder(teeth_sma,
jaw_sma) and profit > 0:
        return 'sell'


    return 'hold'
```

Приложение 7. Модуль удаления инструмента

```
@bot.callback_query_handler(func=lambda call: call.data ==
'delete_instrument')
def delete_instrument_handler(call):
    """
    Обработчик для удаления инструмента.

    Отображает список доступных инструментов для удаления.
    """
    chat_id = call.message.chat.id

    try:
        # Отправляем сообщение о начале обработки
        send_or_edit_message(chat_id, '*Обработка
запроса*\n\nПолучаем список инструментов...')

        # Получаем список всех инструментов через API-клиент
        instruments = instruments_client.get_all_instruments()

        if not instruments:
            send_or_edit_message(chat_id, '*Удаление
инструмента*\n\nУ вас нет активных инструментов')
        else:
            inline_keyboard = types.InlineKeyboardMarkup()
            for instrument in instruments:
                ticker = instrument.get('ticker')
                button = types.InlineKeyboardButton(text=f"
{ticker}", callback_data=f'ticker_{ticker}')
                inline_keyboard.add(button)

            send_or_edit_message(
                chat_id,
                '*Удаление инструмента*\n\nВыберите инструмент
для удаления:',
                reply_markup=inline_keyboard
            )

    except Exception as e:
        send_or_edit_message(chat_id, f'*Ошибка при получении
списка инструментов*\n\n`{str(e)}`')


@bot.callback_query_handler(func=lambda call:
call.data.startswith('ticker_'))
def delete_ticker_step(call):
    """
    Обработчик для удаления выбранного инструмента.

    Удаляет инструмент по выбранному тикеру.
    """
    chat_id = call.message.chat.id
    ticker = call.data.replace('ticker_', '')
```

```
    try:
        # Отправляем сообщение о начале обработки
        send_or_edit_message(chat_id, f'*Обработка
запроса*\n\nУдаляем инструмент `{ticker}`...')

        # Удаляем инструмент через API-клиент
        instruments_client.delete_instrument(ticker)
        send_or_edit_message(chat_id, f'*Успешно*\n\nИнструмент
`{ticker}` успешно удален')

    except Exception as e:
        send_or_edit_message(chat_id, f'*Ошибка при удалении
инструмента*\n\n`{str(e)}`')
```

Приложение 8. Программный код обработчиков настройки сигнала Alligator

```
from telebot import types
from app.client.api.signals_client import SignalsApiClient
from app.client.bot.bot import bot
from app.client.handlers.utils.message_utils import
send_or_edit_message

signals_client = SignalsApiClient()

user_alligator_data = {}


@bot.callback_query_handler(func=lambda call: call.data ==
'signal_alligator')
def alligator_on(call):
    """
    Обработчик для настройки сигнала Alligator.

    Запрашивает у пользователя параметры сигнала Alligator.
    """
    chat_id = call.message.chat.id

    # Получаем текущие настройки Alligator
    current_settings = signals_client.get_signal_alligator()

    if current_settings:
        jaw_period = current_settings.get('jaw_period', 13)
        jaw_shift = current_settings.get('jaw_shift', 8)
        teeth_period = current_settings.get('teeth_period', 8)
        teeth_shift = current_settings.get('teeth_shift', 5)
        lips_period = current_settings.get('lips_period', 5)
        lips_shift = current_settings.get('lips_shift', 3)

        send_or_edit_message(
            chat_id,
            f'👆 *Текущие настройки Alligator*\n\n'
            f'• *Челюсти* - Период: `{jaw_period}`, Смещение:
`{jaw_shift}`\n'
            f'• *Зубы* - Период: `{teeth_period}`, Смещение:
`{teeth_shift}`\n'
            f'• *Губы* - Период: `{lips_period}`, Смещение:
`{lips_shift}`'
        )

    msg = send_or_edit_message(chat_id, "👆 *Настройка
Alligator*\n\nВведите период для челюстей (Jaw):")
    bot.register_next_step_handler(msg,
get_alligator_jaw_period)
```

```
def validate_number(value, min_value=None, max_value=None):
    """
    Проверка, что значение является целым числом с возможной
дополнительной проверкой на диапазон.

    Args:
        value: Проверяемое значение
        min_value: Минимальное допустимое значение (опционально)
        max_value: Максимальное допустимое значение
(опционально)

    Returns:
        bool: True, если значение валидно, иначе False
    """
    try:
        num = int(value)

        # Проверка на диапазон
        if min_value is not None and num < min_value:
            return False
        if max_value is not None and num > max_value:
            return False

        return True
    except ValueError:
        return False


def get_alligator_jaw_period(message):
    """
    Обработчик для получения периода челюстей Alligator.

    Сохраняет период челюстей и запрашивает смещение челюстей.
    """
    chat_id = message.chat.id
    jaw_period = message.text

    if not validate_number(jaw_period, min_value=1,
max_value=100):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nПериод для челюстей должен быть целым числом от 1 до
100. Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_jaw_period)
        return

    jaw_period = int(jaw_period)
    user_alligator_data[chat_id] = {'jaw_period': jaw_period}  #
Сохраняем период для челюстей
    msg = send_or_edit_message(chat_id, f"✅ Вы выбрали период
`{jaw_period}` для челюстей.\n\n*Введите смещение для челюстей
(Jaw shift):*")
    bot.register_next_step_handler(msg, get_alligator_jaw_shift)
```

```
def get_alligator_jaw_shift(message):
    """
    Обработчик для получения смещения челюстей Alligator.

    Сохраняет смещение челюстей и запрашивает период зубов.
    """
    chat_id = message.chat.id
    jaw_shift = message.text

    if not validate_number(jaw_shift, min_value=0):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nСмещение для челюстей должно быть неотрицательным
числом. Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_jaw_shift)
        return

    jaw_shift = int(jaw_shift)  # Преобразуем в целое число
    user_alligator_data[chat_id]['jaw_shift'] = jaw_shift  #
Сохраняем смещение для челюстей
    msg = send_or_edit_message(chat_id, f"✅ Вы выбрали смещение
`{jaw_shift}` для челюстей.\n\n*Введите период для зубов
(Teeth):*")
    bot.register_next_step_handler(msg,
get_alligator_teeth_period)


def get_alligator_teeth_period(message):
    """
    Обработчик для получения периода зубов Alligator.

    Сохраняет период зубов и запрашивает смещение зубов.
    """
    chat_id = message.chat.id
    teeth_period = message.text

    if not validate_number(teeth_period, min_value=1,
max_value=100):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nПериод для зубов должен быть целым числом от 1 до 100.
Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_teeth_period)
        return

    teeth_period = int(teeth_period)  # Преобразуем в целое
число
    user_alligator_data[chat_id]['teeth_period'] = teeth_period
# Сохраняем период для зубов
```

```
    msg = send_or_edit_message(chat_id, f"✅ Вы выбрали период
`{teeth_period}` для зубов.\n\n*Введите смещение для зубов
(Teeth shift):*")
    bot.register_next_step_handler(msg,
get_alligator_teeth_shift)


def get_alligator_teeth_shift(message):
    """
    Обработчик для получения смещения зубов Alligator.

    Сохраняет смещение зубов и запрашивает период губ.
    """
    chat_id = message.chat.id
    teeth_shift = message.text

    if not validate_number(teeth_shift, min_value=0):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nСмещение для зубов должно быть неотрицательным числом.
Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_teeth_shift)
        return

    teeth_shift = int(teeth_shift)  # Преобразуем в целое число
    user_alligator_data[chat_id]['teeth_shift'] = teeth_shift  #
Сохраняем смещение для зубов
    msg = send_or_edit_message(chat_id, f"✅ Вы выбрали смещение
`{teeth_shift}` для зубов.\n\n*Введите период для губ (Lips):*")
    bot.register_next_step_handler(msg,
get_alligator_lips_period)


def get_alligator_lips_period(message):
    """
    Обработчик для получения периода губ Alligator.

    Сохраняет период губ и запрашивает смещение губ.
    """
    chat_id = message.chat.id
    lips_period = message.text

    if not validate_number(lips_period, min_value=1,
max_value=100):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nПериод для губ должен быть целым числом от 1 до 100.
Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_lips_period)
        return

    lips_period = int(lips_period)  # Преобразуем в целое число
```

```
    user_alligator_data[chat_id]['lips_period'] = lips_period  #
Сохраняем период для губ
    msg = send_or_edit_message(chat_id, f"✅ Вы выбрали период
`{lips_period}` для губ.\n\n*Введите смещение для губ (Lips
shift):*")
    bot.register_next_step_handler(msg,
get_alligator_lips_shift)


def get_alligator_lips_shift(message):
    """
    Обработчик для получения смещения губ Alligator.

    Сохраняет смещение губ и обновляет настройки сигнала
Alligator.
    """
    chat_id = message.chat.id
    lips_shift = message.text

    if not validate_number(lips_shift, min_value=0):
        msg = send_or_edit_message(chat_id, "❌ *Ошибка
ввода*\n\nСмещение для губ должно быть неотрицательным числом.
Попробуйте снова:")
        bot.register_next_step_handler(msg,
get_alligator_lips_shift)
        return

    lips_shift = int(lips_shift)  # Преобразуем в целое число
    user_alligator_data[chat_id]['lips_shift'] = lips_shift  #
Сохраняем смещение для губ

    # Получаем все введённые параметры
    jaw_period = user_alligator_data[chat_id]['jaw_period']
    jaw_shift = user_alligator_data[chat_id]['jaw_shift']
    teeth_period = user_alligator_data[chat_id]['teeth_period']
    teeth_shift = user_alligator_data[chat_id]['teeth_shift']
    lips_period = user_alligator_data[chat_id]['lips_period']

    try:
        # Обновляем параметры через API-клиент
        result = signals_client.update_signal_alligator(
            jaw_period, jaw_shift, teeth_period, teeth_shift,
lips_period, lips_shift
        )

        # Подтверждение настройки стратегии
        send_or_edit_message(
            chat_id,
            f"✅ *Стратегия Аллигатор настроена успешно*\n\n"
            f"• *Челюсти* - Период: `{jaw_period}`, Смещение:
`{jaw_shift}`\n"
```

```
            f"• *Зубы* - Период: `{teeth_period}`, Смещение:
`{teeth_shift}`\n"
            f"• *Губы* - Период: `{lips_period}`, Смещение:
`{lips_shift}`"
        )

    except Exception as e:
        send_or_edit_message(chat_id, f"❌ *Ошибка при
обновлении настроек Alligator*\n\n`{str(e)}`")

    del user_alligator_data[chat_id]
```

Приложение 9. Программный код построения графика для сигнала SMA

```
def plot_sma(chat_id, df):
    """
    Функция для построения свечного графика цены и сигналов SMA, и отправки его в Telegram.

    :param chat_id: Идентификатор чата в Telegram, куда нужно отправить график.
    :param df: DataFrame с колонками 'time', 'open', 'high', 'low', 'close' для построения свечного графика цены.
    """

    fast_sma = store.fast_sma
    slow_sma = store.slow_sma

    # Конвертация столбца 'time' в формат для matplotlib
    df['time'] = pd.to_datetime(df['time'])
    df['time'] = df['time'].map(mdates.date2num)

    fig, ax = plt.subplots(figsize=(14, 7))
    ax.set_title('Stock Price with SMA')
    ax.set_xlabel('Date')
    ax.set_ylabel('Price')

    # Создаем список данных для candlestick_ohlc
    ohlc = df[['time', 'open', 'high', 'low', 'close']].values

    candlestick_ohlc(ax, ohlc, width=0.8, colorup='green', colordown='red')

    # Добавление графиков SMA
    ax.plot(df['time'], fast_sma, label='Fast SMA', color='orange', linestyle='--')
    ax.plot(df['time'], slow_sma, label='Slow SMA', color='blue', linestyle='--')

    # Форматирование дат на оси X
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    fig.autofmt_xdate()

    # Добавление легенды
    ax.legend(loc='upper left')

    # Сохранение графика во временный файл
    file_path = 'sma_candlestick_chart.png'
    plt.savefig(file_path)
    plt.close(fig)

    with open(file_path, 'rb') as photo:
        bot.send_photo(chat_id, photo)
```

Приложение 10. Программный код функции для визуализации статистики

```
def statistics_graph(buy, margin, chat_id):
    """
    Функция для формирования статистики по покупкам и продажам.

    Args:
        buy (list): Список покупок.
        margin (list): Список продаж.
        chat_id (int): Id чата, в который будет отправлена
статистика.
    """
    if len(buy) > 0:
        # Преобразуем данные в DataFrame для удобства работы
        buy_df = pd.DataFrame(buy, columns=['id', 'price',
'ticker', 'signal', 'time', 'chat_id'])
        buy_df['time'] = pd.to_datetime(buy_df['time'],
format='%d-%m-%Y %H:%M')

        # 1. Гистограмма покупок тикеров
        plt.figure(figsize=(14, 9))
        if buy_df['ticker'].nunique() > 1:
            sns.countplot(data=buy_df, x='ticker',
palette='Blues')
        else:
            sns.countplot(data=buy_df, x='ticker',
palette='Blues', width=0.1)
        plt.title('Количество покупок по тикерам')
        plt.xlabel('Тикер', labelpad=20)
        plt.ylabel('Количество покупок', labelpad=20)
        plt.xticks(rotation=0, ha='center', va='top')
        plt.gca().tick_params(axis='x', pad=10)
        plt.subplots_adjust(bottom=0.2)
        file_path = 'buy_ticker_histogram.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 2. Гистограмма покупок по сигналам
        plt.figure(figsize=(14, 9))
        signals = buy_df['signal']
        if signals.nunique() > 1:
            sns.countplot(x=signals, palette='Greens')
        else:
            sns.countplot(x=signals, palette='Greens',
width=0.1)
        plt.title('Количество покупок по сигналам')
        plt.xlabel('Сигнал', labelpad=20)
        plt.ylabel('Количество покупок', labelpad=20)
        plt.xticks(rotation=0, ha='center', va='top')
```

```
        plt.gca().tick_params(axis='x', pad=10)
        plt.subplots_adjust(bottom=0.2)
        file_path = 'buy_signal_histogram.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 3. Линейный график покупок по дням или часам
        buy_per_day =
buy_df.groupby(buy_df['time'].dt.strftime('%d-%m-%Y')).size()

        plt.figure(figsize=(14, 9))
        if len(buy_per_day) > 1:
            buy_per_day.plot(kind='line', marker='o',
color='orange')
            plt.title('Количество покупок по дням')
            plt.xlabel('Дата', labelpad=20)
            plt.ylabel('Количество покупок', labelpad=20)
            plt.xticks(rotation=0, ha='center', va='top')
            plt.gca().tick_params(axis='x', pad=10)
            plt.subplots_adjust(bottom=0.2)
        else:
            buy_per_hour =
buy_df.groupby(buy_df['time'].dt.strftime('%H:%M')).size()
            buy_per_hour.plot(kind='bar', color='orange',
width=0.1)
            plt.title('Количество покупок по часам')
            plt.xlabel('Время', labelpad=20)
            plt.ylabel('Количество покупок', labelpad=20)
            plt.xticks(rotation=0, ha='center', va='top')
            plt.gca().tick_params(axis='x', pad=10)
            plt.subplots_adjust(bottom=0.2)

        plt.xticks(rotation=0, ha='center', va='top')
        plt.gca().tick_params(axis='x', pad=10)
        plt.subplots_adjust(bottom=0.2)
        file_path = 'buy_time_graph.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 4. Общая сумма покупок
        total_buy_amount = buy_df['price'].sum()
        bot.send_message(chat_id, f"Общая сумма покупок:
{total_buy_amount} руб.")

    if len(margin) > 0:
        margin_df = pd.DataFrame(margin, columns=['id',
'margin', 'ticker', 'signal', 'time', 'chat_id'])
```

```
        margin_df['time'] = pd.to_datetime(margin_df['time'],
format='%d-%m-%Y %H:%M')

        # 5. Круговой график маржи
        positive_margin = margin_df[margin_df['margin'] >
0].shape[0]
        negative_margin = margin_df[margin_df['margin'] <
0].shape[0]
        data = [positive_margin, negative_margin]
        labels = ['Положительная маржа', 'Отрицательная маржа']

        plt.figure(figsize=(12, 7))
        plt.pie([v for v in data if v > 0], labels=[l for v, l
in zip(data, labels) if v > 0],
                autopct='%1.1f%%', colors=['orange',
'deepskyblue'])
        plt.title('Соотношение положительной и отрицательной
маржи')
        file_path = 'margin_pie_chart.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 6. Гистограмма маржи по тикерам (вверх положительная,
вниз отрицательная)
        margin_df_ticker = margin_df.copy()
        margin_df_ticker['margin_type'] = ['Положительная' if m
> 0 else 'Отрицательная' for m in margin_df_ticker['margin']]
        margin_ticker =
margin_df_ticker.pivot_table(index='ticker',
columns='margin_type', aggfunc='size', fill_value=0)


        columns_mapping = {}
        if 'Положительная' in margin_ticker.columns:
            columns_mapping['Положительная'] = 'Положительная'
        if 'Отрицательная' in margin_ticker.columns:
            columns_mapping['Отрицательная'] = 'Отрицательная'

        margin_ticker =
margin_ticker[list(columns_mapping.keys())]  # Убираем столбцы,
которых нет

        margin_ticker_long =
margin_ticker.reset_index().melt(id_vars='ticker',
var_name='margin_type', value_name='count')

        # Построение графика
        plt.figure(figsize=(20, 12))
        negative_counts =
margin_ticker_long[margin_ticker_long['margin_type'] ==
```

```
'Отрицательная']['count'] if 'Отрицательная' in
margin_ticker_long['margin_type'].values else []
        positive_counts =
margin_ticker_long[margin_ticker_long['margin_type'] ==
'Положительная']['count'] if 'Положительная' in
margin_ticker_long['margin_type'].values else []

        bar_width = 0.1
        x = range(len(margin_ticker_long['ticker'].unique()))

        # Количество уникальных тикеров
        num_tickers = len(margin_ticker_long['ticker'].unique())

        if len(positive_counts) > 0:
            plt.bar(x, positive_counts, width=bar_width,
color='green', label='Положительная', align='center')

        if len(negative_counts) > 0:
            plt.bar(x, -negative_counts, width=bar_width,
color='red', label='Отрицательная', align='center')

        # Настройка меток и заголовка
        plt.title('Количество продаж по тикерам')
        plt.xlabel('Тикер', labelpad=20)
        plt.ylabel('Количество продаж')
        plt.xticks(x, margin_ticker_long['ticker'].unique(),
rotation=0)
        plt.axhline(0, color='black', linewidth=0.8)  # Линия по
оси Y на нуле
        plt.legend()

        plt.xlim(-0.5, num_tickers - 0.5)

        file_path = 'margin_ticker_histogram.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 7. Гистограмма маржи по сигналам
        margin_df['margin_type'] = ['Положительная' if m > 0
else 'Отрицательная' for m in margin_df['margin']]
        margin_signal = margin_df.pivot_table(index='signal',
columns='margin_type', aggfunc='size', fill_value=0)

        columns_mapping = {}
        if 'Положительная' in margin_signal.columns:
            columns_mapping['Положительная'] = 'Положительная'
        if 'Отрицательная' in margin_signal.columns:
            columns_mapping['Отрицательная'] = 'Отрицательная'
```

```
        margin_signal =
margin_signal[list(columns_mapping.keys())]

        margin_signal_grouped_long =
margin_signal.reset_index().melt(id_vars='signal',
var_name='margin_type', value_name='count')

        plt.figure(figsize=(20, 12))
        # Массив для отрицательных значений
        negative_signal_counts =
margin_signal_grouped_long[margin_signal_grouped_long['margin_ty
pe'] == 'Отрицательная']['count'] if 'Отрицательная' in
margin_signal_grouped_long['margin_type'].values else []
        positive_signal_counts =
margin_signal_grouped_long[margin_signal_grouped_long['margin_ty
pe'] == 'Положительная']['count'] if 'Положительная' in
margin_signal_grouped_long['margin_type'].values else []

        # Установка ширины баров
        bar_width = 0.1
        x =
range(len(margin_signal_grouped_long['signal'].unique()))

        # Уникальные значения сигнала
        num_signals =
len(margin_signal_grouped_long['signal'].unique())

        # Положительные бары, если они существуют
        if len(positive_signal_counts) > 0:
            plt.bar(x, positive_signal_counts, width=bar_width,
color='green', label='Положительная', align='center')

        # Отрицательные бары, если они существуют, с
отрицательным значением для направления вниз
        if len(negative_signal_counts) > 0:
            plt.bar(x, -negative_signal_counts, width=bar_width,
color='red', label='Отрицательная', align='center')

        # Настройка меток и заголовка
        plt.title('Количество продаж по сигналам')
        plt.xlabel('Сигнал', labelpad=20
        plt.ylabel('Количество продаж')
        plt.xticks(x,
margin_signal_grouped_long['signal'].unique(), rotation=0)
        plt.axhline(0, color='black', linewidth=0.8)  # Линия по
оси Y на нуле
        plt.legend()
        plt.xlim(-0.5, num_signals - 0.5)

        file_path = 'margin_signal_histogram.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
```

```
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 8. Линейный график продаж по дням или часам
        margin_per_day =
margin_df.groupby(margin_df['time'].dt.strftime('%d-%m-
%Y')).size()
        plt.figure(figsize=(14, 9))
        if len(margin_per_day) > 1:
            margin_per_day.plot(kind='line', marker='o',
color='purple')
            plt.title('Количество продаж по дням')
            plt.xlabel('Дата', labelpad=20)
            plt.ylabel('Количество продаж', labelpad=20)
            plt.xticks(rotation=0, ha='center', va='top')
            plt.gca().tick_params(axis='x', pad=10)
            plt.subplots_adjust(bottom=0.2)
        else:
            margin_per_hour =
margin_df.groupby(margin_df['time'].dt.strftime('%H:%M')).size()
            margin_per_hour.plot(kind='bar', color='purple',
width=0.1)
            plt.title('Количество продаж по часам')
            plt.xlabel('Время', labelpad=20)
            plt.ylabel('Количество продаж', labelpad=20)
            plt.xticks(rotation=0, ha='center', va='top')
        plt.gca().tick_params(axis='x', pad=10)
        plt.subplots_adjust(bottom=0.2)
        plt.xticks(rotation=0)
        file_path = 'sell_time_graph.png'
        plt.savefig(file_path)
        plt.close()
        with open(file_path, 'rb') as photo:
            bot.send_photo(chat_id, photo)
        os.remove(file_path)

        # 9. Общая сумма маржи
        total_margin_percent = margin_df['margin'].sum()
        bot.send_message(chat_id, f"Общая сумма маржи:
{total_margin_percent}%")
```