ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

**РУКОВОДСТВО РАЗРАБОТЧИКА**

**к проекту по дисциплине**

**«Проектный семинар “Python в науке о данных”»**

Студенты группы БИВ248:

• Леснов Илья Андреевич, ialesnov@edu.hse.ru, 89289719490

• Иевлев Николай Евгеньевич, neievlev@edu.hse.ru, 89066087088

• Израйлит Александр Владимирович, avizrailit@edu.hse.ru, 89156172477

Тема работы:

Программный бот для отслеживания аномалий котировок криптовалют

Руководитель:

Ролич Алексей Юрьевич

**1. Введение**

Данный документ содержит техническое описание проекта, его архитектуры, модулей и ключевых функций. Он предназначен для разработчиков, которые хотят понять, как устроен код, или внести в него изменения.

**2. Технологический стек и описание библиотек**

Проект построен на языке Python 3.10 и использует следующие ключевые библиотеки, управляемые через Conda:

* **Python 3.10**: Основа проекта. Используется функциональный подход, без классов и ООП.
* **Conda**: Менеджер пакетов и окружений. Используется для создания изолированной и воспроизводимой среды выполнения через файл environment.yml, что гарантирует идентичность зависимостей у всех разработчиков.
* **Pandas**: Основной инструмент для работы с данными. Применяется для:
  + Создания и хранения временных рядов цен в виде DataFrame.
  + Удобной фильтрации и выборки данных по символам (history\_df[history\_df['symbol'] == symbol]).
  + Быстрых статистических вычислений (.mean(), .std()).
  + Сохранения отчетов в CSV (.to\_csv()).
* **NumPy**: Используется неявно через Pandas для выполнения быстрых математических операций над массивами данных.
* **Matplotlib**: Библиотека для научной визуализации. Применяется для:
  + Создания объекта графика (Figure) и осей (Axes).
  + Построения линейных графиков истории цен (ax.plot()).
  + Интеграции графика в интерфейс Tkinter с помощью matplotlib.backends.backend\_tkagg.FigureCanvasTkAgg.
  + Сохранения графиков в файлы различных форматов (figure.savefig()).
* **CCXT** (CryptoCurrency eXchange Trading Library): Мощная унифицированная библиотека для работы с API более чем 100 криптовалютных бирж.
  + **Назначение:** Абстрагирует разработчика от деталей реализации API каждой конкретной биржи.
  + **Использование:** ccxt.binance() (или другая биржа) для создания объекта-коннектора, exchange.fetch\_tickers() для получения данных.
* **Tkinter (ttk)**: Стандартная библиотека Python для создания графических интерфейсов пользователя (GUI).
  + **Назначение:** Построение окон, фреймов, кнопок, таблиц и других элементов управления.
  + **Использование:** ttk.Treeview для таблиц, ttk.Button для кнопок, root.after() для организации неблокирующего основного цикла приложения.
* **ConfigParser**: Встроенная библиотека для работы с .ini файлами. Используется для чтения и парсинга файла config.ini.

**3. Архитектура проекта**

Проект имеет модульную структуру с четким разделением ответственности. Корневая директория — Work.

* Work/
  + Output/: Хранит генерируемые данные (лог аномалий).
  + Graphics/: Место для сохранения графиков, сгенерированных пользователем.
  + Library/: **Бизнес-логика.** Переиспользуемые модули, не зависящие от интерфейса.
    - api\_handler.py: Взаимодействие с внешним API.
    - data\_analyzer.py: Алгоритм анализа данных.
  + Scripts/: **Логика приложения.** Модули, отвечающие за запуск, конфигурацию и UI.
    - config\_manager.py: Чтение конфигурации.
    - ui\_manager.py: Построение и управление GUI.
    - main.py: Точка входа, оркестратор.
  + config.ini: Файл настроек.
  + environment.yml: Файл для воспроизведения Conda-окружения.

**4. Взаимодействие модулей (Data Flow)**

Поток данных и вызовов в приложении организован централизованно через main.py, который выступает в роли дирижера.

**Схема взаимодействия:** config.ini → config\_manager.py → main.py → (api\_handler.py → data\_analyzer.py → ui\_manager.py)

**Шаг 1: Инициализация (в main.main())**

1. main.py вызывает config\_manager.load\_config('../config.ini'), чтобы прочитать настройки.
2. Получив конфигурацию, main.py вызывает api\_handler.connect\_to\_exchange(config['api']['exchange']) для установки соединения с биржей.
3. Затем main.py вызывает функции из ui\_manager (create\_main\_window, create\_widgets и др.), передавая им параметры из конфига для построения интерфейса.
4. main.py запускает основной цикл, вызывая root.after(..., main\_update\_cycle).

**Шаг 2: Основной цикл обновления (в main.main\_update\_cycle())**

Это сердце приложения, где модули работают в связке:

1. **Получение данных:** main.py вызывает api\_handler.fetch\_tickers(...).
   * **Вход:** Объект биржи, список символов.
   * **Выход:** pandas.DataFrame со свежими ценами.
2. **Анализ данных:** main.py в цикле для каждой полученной цены вызывает data\_analyzer.find\_anomalies(...).
   * **Вход:** Вся историческая таблица history\_df, текущая цена, символ и параметры анализа (окно, порог).
   * **Выход:** Словарь с деталями аномалии или None.
3. **Обновление UI:** Если найдена аномалия, main.py вызывает ui\_manager.update\_anomaly\_log(...). В любом случае вызывается ui\_manager.update\_prices\_table(...).
   * **Вход:** Объект виджета (например, ttk.Treeview) и данные для отображения.
   * **Выход:** Визуальное обновление интерфейса.

**Шаг 3: Реакция на действия пользователя (в main)**

1. **Клик по таблице:** Событие <<TreeviewSelect>> вызывает функцию main.on\_symbol\_select.
   * Эта функция определяет выбранный символ и вызывает ui\_manager.update\_graph(...), передавая ему оси графика (ax), холст (canvas), исторические данные и выбранный символ. ui\_manager перерисовывает график.
2. **Нажатие "Сохранить график"**: Кнопка вызывает функцию main.save\_graph\_to\_file.
   * Эта функция получает объект Figure из app\_state и использует его метод .savefig() для сохранения изображения в папку Graphics/.

**5. Описание ключевых функций модулей**

**Library/api\_handler.py**

Отвечает за получение данных с бирж.

* connect\_to\_exchange(exchange\_name): Инициализирует и возвращает объект биржи ccxt.
* fetch\_tickers(exchange, symbols): Запрашивает актуальные цены для списка символов. Обрабатывает ошибки сети/API и возвращает pandas.DataFrame со столбцами ['timestamp', 'symbol', 'price'].

**Library/data\_analyzer.py**

Реализует алгоритм выявления аномалий.

* find\_anomalies(history\_df, current\_price, symbol, window, threshold): Основная аналитическая функция.
  + **Алгоритм:** Рассчитывает среднее и стандартное отклонение на основе последних window точек из **прошлой истории** (history\_df).
  + Сравнивает **текущую цену** (current\_price) с вычисленным "коридором нормы" (mean ± std \* threshold).
  + Возвращает словарь с деталями, если найдена аномалия, иначе None.

**Scripts/config\_manager.py**

Отвечает за парсинг config.ini.

* load\_config(path): Читает .ini файл, преобразует значения в нужные типы (int, float, list) и возвращает в виде словаря.

**Scripts/ui\_manager.py**

Отвечает за построение и обновление GUI на Tkinter.

* **Структура:** Функции разделены на create\_\* (вызываются один раз для построения макета) и update\_\* (вызываются для обновления данных в виджетах).
* **Ключевые виджеты:** ttk.Treeview для таблиц, FigureCanvasTkAgg для интеграции Matplotlib в окно Tkinter.
* **Обновление:** Функции update\_prices\_table (с тегами для цветовой индикации), update\_anomaly\_log, update\_graph.

**Scripts/main.py**

Точка входа и главный координатор приложения.

* **Глобальное состояние:** Используется словарь app\_state для хранения общих данных (объекты окна, виджетов, история цен).
* **Инициализация:** В функции main() происходит загрузка конфига, создание директорий, подключение к бирже и построение GUI.
* **Основной цикл:** main\_update\_cycle() — рекурсивная функция, вызываемая через root.after(). Она выполняет один цикл (запрос данных -> анализ -> обновление UI) и планирует свой следующий вызов, не блокируя GUI.
* **Обработка событий:** Привязывает функции-обработчики к событиям виджетов:
  + on\_symbol\_select: Вызывается при клике на строку в таблице для обновления графика.
  + save\_graph\_to\_file: Вызывается при нажатии на кнопку "Сохранить график".

**6. Порядок запуска и отладки**

1. **Запуск:** cd /path/to/Work/Scripts и затем python main.py. Убедитесь, что Conda-окружение crypto-anomaly-bot активировано.
2. **Отладка:** Каждый модуль (api\_handler.py, data\_analyzer.py и др.) содержит блок if \_\_name\_\_ == '\_\_main\_\_':. Это позволяет запускать каждый файл по отдельности для изолированного тестирования его функциональности. Например, python data\_analyzer.py запустит только тесты для алгоритма, не поднимая весь GUI. Это критически важный инструмент для проверки логики каждого компонента в отдельности.