# **Technical Analysis: Arbitrage Bot**

## **1. Project Overview**

* **Name/Goal** *Building a cryptocurrency arbitrage bot that detects and exploits price discrepancies across multiple centralized exchanges, using custom-built,CCXT-based and CoinGecko-based exchange connectors.*
* **Primary Focus**
  + **Fetch and unify market data** (price, volume, order books) from selected exchanges.
  + **Detect arbitrage opportunities** (simple cross-exchange, possibly triangular).
  + **(Optional) Automate partial or full trade execution** if time permits, or outline it as a future enhancement.

## **2. Exchange Selection and Connectivity**

### **2.1 Selected Exchanges**

* **Primary Custom Connectors**
  + **Binance**
    - Largest liquidity, well-documented REST and WebSocket APIs, standard HMAC authentication.
  + **KuCoin**
    - Decent liquidity, moderate fees, known API quirks
* **Additional Exchanges via CCXT**
  + **Kraken, Coinbase, etc.**
    - Will be accessed through CCXT to demonstrate broad coverage without individually re-implementing every exchange API.

This approach should show deep **technical mastery** in custom connector design (for Binance/KuCoin) while **leveraging** CCXT’s reliability for additional exchanges.

## **3. APIs, Libraries & Tools**

### **3.1 APIs**

* **Binance REST API**
  + Endpoints for order books, trades, account data, etc.
  + Authentication: API key + secret, HMAC signature with each request.
  + Rate limits: ~1200 requests/minute (depending on account tier).
* **KuCoin REST API**
  + Similar structure but with slightly different endpoint naming.
  + Also HMAC + additional passphrase for certain calls.
* **CCXT**
  + Provides a unifying interface for other major exchanges (Kraken, Coinbase Pro, Huobi, etc.).
  + Eliminates the need to implement each exchange’s authentication & REST calls.
* **CoinGecko** 
  + Will be used for first phase as an introduction to working with APIs
  + Has good documentation and guides how to get started

### **3.2 Python Libraries**

* **Requests/httpx** or built-in aiohttp
  + For REST calls if implementing connectors asynchronously.
* **CCXT**
  + For multi-exchange coverage beyond the custom-coded connectors.
* **Pandas/Numpy**
  + Data manipulation, calculations for spreads, logs, etc.
* **Asyncio** (Potential)
  + Parallelizing data fetching from multiple exchanges to reduce latency.
* **Json**
  + Manipulating with API keys and requests
* **Warnings**
  + For clearer error handling

### **3.3 Other Tools**

* **Version Control**
  + Git + GitHub (GitLab) for code repository and collaboration.
* **CI/CD (Optional)**
  + Basic testing pipeline to ensure connectors remain functional when exchange APIs update.
* **Logging & Monitoring**
  + Python’s logging module for runtime debugging and error reporting.

## **4. System Architecture**

### **4.1 Core Components**

1. **Custom Connector: Binance**
   * *Implements REST calls for market data, order placement/cancellation, balances.*
   * *Handles signature generation, timestamps, error handling, rate limiting.*
2. **Custom Connector: KuCoin**
   * *Similar scope, but adjusted for KuCoin’s request signing, endpoints, and partial fill logic.*
3. **CCXT Connector**
   * For other exchanges we want to monitor or potentially trade on, but for which we haven’t built custom connectors.
   * Minimizes overhead while allowing broad coverage.
4. **Arbitrage Detection Module**
   * Fetches best bid/ask across all connected exchanges, calculates spreads.
   * Compares potential profit vs. fees/slippage thresholds.
   * If a profitable spread is found, logs it or triggers the next step (e.g., a trade).
5. **Execution Logic (Optional / Future)**
   * If time allows, place real or paper orders.
   * For demonstration, we might do partial automation (e.g., a “confirm trade”)

## **5. Phased Implementation & Milestones**

1. **Phase 1: Basic Market Data & Min/Max Detection (~2–3 weeks)**
   * **Overview**: Building a basic arbitrage program, which connects to coingecko API, fetches market data and finds where the max and min value of a single coin on different exchanges is, It will highlight it and this is the possible arbitrage (This strategy doesn’t take into account fees for the transactions which in some cases may be higher than the possible profit, which would make the transaction useless)
   * **Coding**: We will aim to build a nice and reliable code base, easy to read and modify. As this Is only a starting point of the project, many things may not be usable later and will be replaced by better versions more optimized for our purpose. We will define basic packages and Classes needed for the simple arbitrage program.
   * **Aim**: This phase is the first step, an introduction so to say, to the arbitrage opportunity finding. Its purpose is to become comfortable with fetching data using APIs and then using it in meaningful ways. It is not meant to be the final version of the project, which we will improve in later phases of our project.
2. **Phase 2: Improved Market Data & Min/Max Detection (~4–6 weeks after phase 1)**
   * **Overview:** Adopting the hybrid approach: we will continue using CoinGecko for broader market data but also incorporate CCXT for easy access to multiple centralized exchanges. In parallel, we will implement our own custom classes to interact directly with the Binance and KuCoin REST APIs. This allows us to demonstrate a deeper technical understanding of exchange-specific logic while leveraging CCXT to cover additional exchanges with less overhead.
   * **Coding:** Custom Connectors: Build dedicated modules/classes for Binance and KuCoin, handling REST endpoints, authentication (API key, secret, passphrase), and error handling. CCXT Integration: Use CCXT to fetch data or place orders on other exchanges, providing broader coverage without rewriting every single API connector. Data Aggregation & Comparison: Collect price/order book data from both custom connectors and CCXT-based connectors, then unify them into a single structure for arbitrage analysis.
   * **Aim:** By the end of this phase, we aim to have two robust, fully custom connectors (Binance and KuCoin), plus multi-exchange coverage via CCXT. This hybrid model demonstrates both low-level mastery of exchange APIs and practical efficiency for rapid expansion to new exchanges.
3. **Phase 3: Refine Code + Simplified Trading (Optional) (~4–6 weeks after phase 2)** 
   * **Overview:** Refine the existing connectors, introduce simplified trading features (such as paper trading or small-scale live trades), and handle more sophisticated issues like fees, slippage, and partial fills. This phase marks the transition from mere opportunity detection to more realistic scenarios.
   * **Coding:** 
     1. Connector Enhancements: Improve error handling (e.g., rate limits, timeouts), handle partial fills more gracefully, incorporate security best practices for API keys.
     2. Paper Trading Logic: Place simulated buy/sell orders, track PnL over time. Log each trade to measure performance and spot inefficiencies.
     3. Risk & Fees: Start accounting for trading fees and potential slippage, implement threshold checks so we only pursue trades with sufficient estimated profit.
   * **Aim:** Build a minimally viable trading system with proper logging, error management, and an understanding of real-world constraints. By the end of Phase 3, we should have a functional, if simplified, arbitrage bot that can simulate or partially execute trades under realistic conditions—setting the stage for more advanced automation or strategy tweaks in the future.

**Potential Future Improvements**

* 1. **Full Automation**: Add robust scheduling, monitor the bot 24/7, handle edge cases around API downtime, partial fills, or liquidity shortfalls.
  2. **Machine Learning**: Predictive models for timing entries or dynamically adjusting threshold for arbitrage profitability.

## **6. Strategic & Technical Rationale**

### **6.1 Why Hybrid Connectors?**

* **Educational Depth**: Implementing at least two full connectors demonstrates proficiency in REST API interaction, authentication, and error-handling.
* **Practical Efficiency**: CCXT provides immediate access to many exchanges without needing to reinvent each wheel.
* **Maintainability**: The custom connectors can be thoroughly documented in the thesis, while CCXT covers the rest with minimal overhead.

### **6.2 Trade Execution & Risk**

* For now, the **execution** portion is optional:
  + Implementation detail often requires elaborate testing and handling real user funds.
* If time allows, we can implement a limited **real-trade** scenario on a testnet or with small capital.

## **7. References and External Components**

1. **Exchange API Docs**
   * [Binance](https://developers.binance.com/docs/binance-spot-api-docs)
   * [KuCoin](https://docs.kucoin.com/)
2. **CCXT**
   * [GitHub Repository](https://github.com/ccxt/ccxt)
   * [Official Documentation](https://docs.ccxt.com/en/latest/)
3. **Python Libraries**
   * Requests (or aiohttp for async), pandas, numpy, warnings, datetime requests, time, json.
4. **Project/Time Management**
   * GitHub Project.

## **8. Grand total timeline**

* **Phase 1:** ~2–3 weeks **End date**: ~22.3.2025–29.3.2025
* **Phase 2**: ~4–6 weeks
* **Phase 3**: ~4–6 weeks
* **Overall**: ~10–15 weeks

## **9. Conclusion**

This **hybrid approach** provides:

* **Technical originality** (by building custom connectors for major exchanges)
* **Breadth** (via CCXT for additional coverage)
* **Expandability** (option to integrate DEXs or advanced automation later)

It balances **complexity** with **practical scope**, ensuring we can **showcase** lower-level API integration skills *and* higher-level arbitrage logic *without* getting slowed down in implementing every exchange from scratch.

### **Final Note**

We use this analysis as a **living document**—update it if we decide to add or remove specific exchanges, or if our timeframe changes.