High-Level Design Document

1. Introduction

1.1 **Purpose** The purpose of this High-Level Design Document is to provide an overview of the architecture and components of the Cryptocurrency Arbitrage Web Application. This document will guide the development team in building a robust and scalable application that identifies and exploits cryptocurrency arbitrage opportunities.

1.2 **Scope** The document covers the overall architecture, data flow, key components, and interactions between the frontend, backend, database, and external services.

1.3 **Overview** The web application allows users to select cryptocurrencies and exchanges, set investment parameters, view arbitrage opportunities, and execute trades. It is built using ReactJS for the frontend, Flask for the backend, MySQL for the database, and Redis for caching and event handling.

2. Architecture Overview

The architecture of the application is divided into several layers:

1. **Presentation Layer (Frontend)**
    - **Technology**: ReactJS, Tailwind CSS
    - **Responsibilities**: User interface, user input validation, displaying data and opportunities
2. **Business Logic Layer (Backend)**
    - **Technology**: Flask, Python
    - **Responsibilities**: Handling API requests, processing data, detecting arbitrage opportunities, managing trade execution
3. **Data Layer (Database)**
    - **Technology**: MySQL
    - **Responsibilities**: Storing user settings, cryptocurrency prices, arbitrage opportunities, and trade records
4. **Caching Layer**
    - **Technology**: Redis
    - **Responsibilities**: Caching real-time data, handling event-driven updates

3. Component Design

3.1 **Frontend Components**

1. **CryptocurrencySelection**
    - Allows users to select up to 5 cryptocurrencies from a dropdown menu
2. **ExchangeSelection**
    - Allows users to select up to 5 exchanges from a dropdown menu
3. **InvestmentParameters**
    - Input fields for investment amount, maximum open positions, risk level
    - Toggles for trade take profit and trade stop loss
    - Input fields for take profit and stop loss percentages
4. **OrderTypeSelection**

    - Radio buttons for selecting order type (Auto, Market, Two Leg Entry)
5. **CreateBotButton**
    - Button to submit the configuration and create the arbitrage bot
6. **OpportunitiesDisplay**
    - Table to display detected arbitrage opportunities with details like entry price, potential profit, and risk
7. **TradeExecutionStatus**
    - Display for the status of executed trades with details like portfolio, P/L, ROI, and status

3.2 **Backend Components**

1. **API Endpoints**
    - Endpoints to handle requests from the frontend, including bot creation, data retrieval, and trade execution
2. **Data Processing Logic**
    - Logic to collect and process real-time data from exchanges
3. **Arbitrage Detection Algorithm**
    - Algorithm to detect arbitrage opportunities by comparing prices across exchanges
4. **Execution Strategy**
    - Strategy to execute trades efficiently, considering factors like order book depth, liquidity, and slippage
5. **Event Handling**
    - Using Socket.IO to manage real-time updates and notifications

3.3 **Database Schema**

1. **Users**
    - `id`, `username`, `email`, `password`
2. **Settings**
    - `user_id`, `selected_cryptocurrencies`, `selected_exchanges`, `investment_amount`, `max_open_positions`, `risk_level`, `take_profit`, `stop_loss`, `order_type`
3. **Prices**
    - `cryptocurrency`, `exchange`, `price`, `timestamp`
4. **ArbitrageOpportunities**
    - `id`, `cryptocurrency`, `buy_exchange`, `sell_exchange`, `buy_price`, `sell_price`, `potential_profit`, `detected_at`
5. **Trades**
    - `id`, `user_id`, `cryptocurrency`, `buy_exchange`, `sell_exchange`, `buy_price`, `sell_price`, `status`, `executed_at`, `profit_loss`

3.4 **Redis Cache**

- **Real-Time Price Data**
    - Cached to reduce load on the database and improve performance
- **Event Handling**
    - Caching and processing events for real-time trade execution and notifications

4. Data Flow

1. **User Interaction**
    - Users interact with the frontend to select cryptocurrencies, exchanges, and set parameters.
2. **Data Submission**
    - The frontend sends user settings to the backend via API calls.
3. **Real-Time Data Collection**
    - The backend collects real-time price data from exchanges and stores it in the MySQL database and Redis cache.
4. **Arbitrage Detection**
    - The backend processes the data to detect arbitrage opportunities and stores potential opportunities in the database.
5. **Opportunity Display**
    - Detected opportunities are sent to the frontend for display.
6. **Trade Execution**
    - When users decide to execute trades, the backend processes the trade, updates the database, and sends status updates to the frontend.
7. **Real-Time Updates**
    - The system uses Socket.IO to provide real-time updates to the frontend about new opportunities and trade statuses.

5. Integration and Communication

- **Frontend and Backend Communication**
    - Communication via RESTful API endpoints for data submission and retrieval
    - Real-time updates via WebSockets (Socket.IO)
- **Backend and Database Communication**
    - Backend interacts with MySQL for persistent storage and Redis for caching
- **Backend and External Services**
    - Backend collects real-time data from external exchange APIs

6. Security Considerations

- **Data Encryption**
    - Encrypt user data and API keys in transit and at rest
- **Authentication and Authorization**
    - Implement secure authentication mechanisms (e.g., JWT)
    - Role-based access control to ensure only authorized actions
- **API Security**
    - Use CORS to manage API access
    - Rate limiting to prevent abuse

7. Deployment

1. **Frontend**
    - Deploy using a hosting service like Vercel or Netlify
2. **Backend**
    - Deploy on a cloud platform like Heroku or AWS
3. **Database**

    - Set up MySQL on a cloud service like AWS RDS
4. **Cache**
    - Deploy Redis using RedisLabs or AWS ElastiCache

8. Maintenance and Monitoring

- **Regular Updates**
    - Keep the system updated with new features and security patches
- **Monitoring**
    - Use monitoring tools to track performance and detect issues
- **Backup**
    - Regularly back up the database and configurations

By following this High-Level Design Document, the development team can ensure that the Cryptocurrency Arbitrage Web Application is well-structured, scalable, and maintainable, meeting the needs of users and providing a robust platform for arbitrage trading.