Software Requirements Specification (SRS)

1. Introduction

1.1 **Purpose** The purpose of this document is to define the requirements for the development of a Cryptocurrency Arbitrage Web Application. The application aims to identify and exploit arbitrage opportunities across different cryptocurrency exchanges.

1.2 **Scope** The application will allow users to select up to 5 cryptocurrencies and 5 exchanges for trading. The application will collect real-time price data, detect arbitrage opportunities, consider transaction costs, implement risk management strategies, and automate the trading process.

1.3 **Definitions, Acronyms, and Abbreviations**

- **Arbitrage**: The simultaneous purchase and sale of an asset to profit from a difference in the price.
- **API**: Application Programming Interface.
- **UI**: User Interface.
- **CORS**: Cross-Origin Resource Sharing.
- **Socket.IO**: A library for real-time web applications.

1.4 **References**

- Flask Documentation: https://flask.palletsprojects.com/
- ReactJS Documentation: https://reactjs.org/
- MySQL Documentation: https://dev.mysql.com/doc/
- Tailwind CSS Documentation: https://tailwindcss.com/
- Socket.IO Documentation: https://socket.io/

2. Overall Description

2.1 **Product Perspective** The Cryptocurrency Arbitrage Web Application will be a standalone system that integrates multiple services and databases. The front end will be developed using ReactJS and Tailwind CSS, while the backend will be built with Flask. Data will be stored in a MySQL database, and Redis will be used for caching and event handling.

2.2 **Product Functions**

- **User Interface**: Allow users to select cryptocurrencies and exchanges, set investment parameters, and view arbitrage opportunities.
- **Data Collection**: Collect and store real-time price data from multiple exchanges.
- **Arbitrage Detection**: Identify potential arbitrage opportunities by comparing prices across exchanges.
- **Transaction Cost Consideration**: Evaluate arbitrage opportunities considering transaction fees and taxes.
- **Risk Management**: Implement strategies to manage risk.
- **Execution Strategy**: Execute trades efficiently considering market conditions.
- **Monitoring and Automation**: Continuously monitor the market and automate the trading process.

2.3 **User Characteristics** The primary users of this application are cryptocurrency traders and investors with basic knowledge of arbitrage trading and cryptocurrency markets.

2.4 **Constraints**

- The system must handle high volumes of real-time data.
- The application must comply with regulatory considerations for cryptocurrency trading.
- The system should be scalable to accommodate future growth.

2.5 **Assumptions and Dependencies**

- The application assumes that users have valid API keys for the selected exchanges.
- The system depends on the availability and reliability of the selected exchanges' APIs.

3. Specific Requirements

3.1 **Functional Requirements**

3.1.1 **User Interface**

- The user shall be able to select up to 5 cryptocurrencies and 5 exchanges for trading.
- The user shall be able to set investment parameters such as investment amount, max open positions, risk level, take profit percentage, and order type.
- The system shall display detected arbitrage opportunities in a tabular format.

3.1.2 **Data Collection**

- The system shall collect real-time price data for selected cryptocurrencies from multiple exchanges.
- The system shall store collected data in a MySQL database.

3.1.3 **Arbitrage Detection**

- The system shall detect potential arbitrage opportunities by comparing prices for the same cryptocurrency across different exchanges.
- The system shall evaluate arbitrage opportunities considering transaction costs and taxes.

3.1.4 **Risk Management**

- The system shall implement risk management strategies to mitigate potential risks associated with executing arbitrage trades.
- The system shall allow users to set risk levels to manage exposure.

3.1.5 **Execution Strategy**

- The system shall design an execution strategy to efficiently execute arbitrage trades.
- The system shall consider factors such as order book depth, liquidity, and slippage.

3.1.6 **Monitoring and Automation**

- The system shall continuously monitor the market for new arbitrage opportunities.
- The system shall automate the trading process to capitalize on arbitrage opportunities quickly.

3.2 **Non-Functional Requirements**

3.2.1 **Performance**

- The system shall process real-time data with minimal latency.
- The system shall support concurrent users without performance degradation.

3.2.2 **Reliability**

- The system shall be available 99.9% of the time.
- The system shall handle API failures gracefully and retry failed requests.

3.2.3 **Scalability**

- The system shall be scalable to handle increased data volumes and user growth.

3.2.4 **Security**

- The system shall secure user data and API keys using encryption.
- The system shall implement authentication and authorization mechanisms.

3.2.5 **Maintainability**

- The system shall be designed for easy maintenance and updates.
- The codebase shall follow best practices for readability and documentation.

3.3 **Database Requirements**

- The system shall use MySQL for storing cryptocurrency prices and user data.
- The system shall use Redis for caching and event handling.

3.4 **API Requirements**

- The system shall interact with cryptocurrency exchange APIs to fetch real-time data.
- The system shall expose APIs for the frontend to interact with the backend.

3.5 **WebSocket Requirements**

- The system shall use WebSocket for real-time updates and notifications.

4. System Models

4.1 **Use Case Diagrams** (Provide diagrams representing the interaction between users and the system)

4.2 **Sequence Diagrams** (Provide diagrams showing the sequence

of interactions between components of the system)

4. System Models

4.1 Use Case Diagrams

Here is a high-level use case diagram illustrating the main interactions between the user and the system:

```
sql
Copy code
      +---------------------------------+
      |        User                     |
      |---------------------------------|
      | - Select Cryptocurrencies       |
      | - Select Exchanges              |
      | - Set Investment Parameters     |
      | - View Arbitrage Opportunities  |
      | - Execute Trades                |
      +---------------------------------+
                  |
                  |
                  V
      +---------------------------------+
      |        System                   |
      |---------------------------------|
      | - Collect Real-time Data        |
      | - Detect Arbitrage              |
      | - Evaluate Opportunities        |
      | - Manage Risks                  |
      | - Execute Trades                |
      | - Monitor & Automate            |
      +---------------------------------+
```

4.2 **Sequence Diagrams**

Sequence diagram showing the flow of actions when a user sets up an arbitrage bot and executes a trade:

```
sql
Copy code
User     Frontend     Backend     Database     Exchanges
 |           |            |            |              |
 |---Select Cryptos------>|            |              |
 |---Select Exchanges---->|            |              |
 |---Set Parameters------>|            |              |
 |                        |---Save Settings---------->|
 |                        |<---------ACK--------------|
 |                        |            |              |
 |                        |---Fetch Prices----------->|
 |                        |<-----Prices Data----------|
 |                        |            |              |
 |                        |---Detect Arbitrage-------->|
 |                        |<------Opportunities--------|
 |<-----Show Opportunities|            |              |
 |---Execute Trade------->|            |              |
 |                        |---Place Orders----------->|
 |                        |<------Order Status--------|
 |<-----Show Trade Status |            |              |
```

5. User Interface Design

The UI design will follow the example provided in the initial requirements. The main components of the UI will include:

1. **Cryptocurrency and Exchange Selection**
    - Dropdown menus to select up to 5 cryptocurrencies.
    - Dropdown menus to select up to 5 exchanges.
2. **Investment Parameters**
    - Input fields for investment amount, maximum open positions, and risk level.
    - Toggle switches for trade take profit and trade stop loss.
    - Input fields for take profit percentage and stop loss percentage.
3. **Order Type Selection**
    - Radio buttons for selecting order type (Auto, Market, Two Leg Entry).
4. **Create Button**
    - Button to submit the configuration and create the arbitrage bot.
5. **Opportunities Display**
    - Table or list to display detected arbitrage opportunities with details like entry price, potential profit, and risk.
6. **Trade Execution Status**
    - Display the status of executed trades with details like portfolio, P/L, ROI, and status.

6. System Architecture

The system will follow a multi-tier architecture:

1. **Frontend**

    - **Technology**: ReactJS, Tailwind CSS
    - **Components**: Selection forms, parameters input, opportunities table, execution status display
2. **Backend**
    - **Technology**: Flask, Socket.IO
    - **Components**: API endpoints, data processing logic, arbitrage detection algorithm, execution strategy
3. **Database**
    - **Technology**: MySQL
    - **Components**: Tables for storing user settings, cryptocurrency prices, arbitrage opportunities, trade executions
4. **Cache**
    - **Technology**: Redis
    - **Components**: Caching real-time price data, event handling for trade executions

7. Testing Requirements

7.1 **Functional Testing**

- Test each UI component for correct functionality.
- Verify API endpoints return correct data.
- Validate the arbitrage detection algorithm with test data.
- Ensure trades are executed correctly and statuses are updated.

7.2 **Performance Testing**

- Test the system's ability to handle high volumes of data.
- Measure response times for real-time data processing and updates.

7.3 **Security Testing**

- Verify user data is encrypted and secured.
- Test authentication and authorization mechanisms.

7.4 **Usability Testing**

- Ensure the UI is intuitive and user-friendly.
- Validate that all user inputs are correctly handled.

7.5 **Integration Testing**

- Test the interaction between frontend and backend.
- Verify that real-time data is correctly processed and displayed.

8. Deployment

8.1 **Deployment Steps**

1. **Frontend**: Build and deploy the ReactJS application on a hosting service like Vercel or Netlify.
2. **Backend**: Deploy the Flask application on a platform like Heroku or AWS.
3. **Database**: Set up a MySQL database on a cloud service like AWS RDS.
4. **Cache**: Deploy Redis on a service like RedisLabs.

8.2 **Post-Deployment**

- Monitor the system for performance and errors.
- Regularly update the application for new features and improvements.

This SRS document outlines the requirements and steps necessary to build and deploy a Cryptocurrency Arbitrage Web Application, ensuring that all functional, non-functional, and technical aspects are covered.