# Backend Developer Assignment: Crash Game with Crypto API and WebSockets

**Overview**

You are tasked with building a backend for an online "Crash" game called "Crypto Crash." Players bet in USD, which is converted to a cryptocurrency (e.g., BTC or ETH) using real-time prices fetched from a cryptocurrency API. They watch a multiplier increase in real-time and decide when to cash out before the game "crashes." The backend must handle game logic, cryptocurrency transactions (simulated with real-time price integration), and real-time multiplayer updates using WebSockets. This assignment evaluates your skills in game development logic, cryptocurrency integration, and WebSocket implementation.

**Requirements**

**1. Game Logic (Crash Game)**

- **Game Rules**:
  - A game round starts every 10 seconds.
  - Players place bets in USD (e.g., $10), which are converted to a chosen cryptocurrency (e.g., BTC or ETH) based on real-time prices.
  - Once the round starts, a multiplier begins at 1x and increases exponentially over time (e.g., using a formula like multiplier = 1 + (time_elapsed * growth_factor)).
  - The game randomly "crashes" at a multiplier value (e.g., 1.5x, 3x, 10x … 120x) determined by a provably fair algorithm.
  - Players can cash out at any time before the crash, earning their bet (in crypto) multiplied by the current multiplier, converted back to USD for display.
  - If a player does not cash out before the crash, they lose their bet.
  - Game state (bets, cashouts, crash point, player balances) must be tracked and stored.
- **Implementation**:
  - Create API endpoints to:
    - Place a bet in USD, specifying the cryptocurrency for conversion.
    - Cash out during a round.
  - Implement a provably fair crash algorithm:
    - Use a cryptographically secure random number generator to determine the crash point.
    - Ensure the crash point is verifiable (e.g., provide a seed and hash for transparency).

    - Example: Crash point could be derived from crash_point = hash(seed + round_number) % max_crash, where max_crash is a high value (e.g., 100x).
  - Store game round history (e.g., round ID, crash point, player bets, cashouts, outcomes) in a database, including USD and crypto amounts.

**2. Cryptocurrency Integration with Real-Time Price API**

- **Task**:
  - Integrate a public cryptocurrency API (e.g., CoinGecko, CoinMarketCap, Binance API or any free api) to fetch real-time prices for supported cryptocurrencies (e.g., BTC, ETH).
  - Allow players to bet in USD, converting the USD amount to the chosen cryptocurrency at the current market price.
  - Simulate a cryptocurrency wallet system for players, storing balances in crypto.
  - Each bet deducts the equivalent crypto amount from the wallet, and cashouts add the crypto payout to it.
  - Simulate blockchain transactions for each bet and cashout (e.g., log transaction details like sender, receiver, crypto amount, and a mock transaction hash).
  - Provide USD-equivalent values for balances and payouts using the latest crypto price for display purposes.
- **Implementation**:
  - Create API endpoints to:
    - Check a player's wallet balance (in crypto and USD equivalent).
    - Place a bet in USD, converting to crypto.
    - Process cash out winnings (add crypto to balance, return USD equivalent).
  - Fetch real-time crypto prices:
    - Cache prices for 10 seconds to avoid rate limits, but ensure conversions use the price at the time of the bet.
  - Conversion logic:
    - Example: Player bets $10 with BTC selected, and BTC price is $60,000. Convert: $10 / $60,000 = 0.00016667 BTC.
    - Cashout at 2x multiplier: 0.00016667 BTC * 2 = 0.00033334 BTC, converted back to USD for display (0.00033334 * $60,000 = $20).
  - Store transaction logs in a database with fields: player_id, usd_amount, crypto_amount, currency, transaction_type (bet/cashout), transaction_hash (mock), price_at_time (USD per crypto), timestamp.

    - Ensure atomicity in balance updates using database transactions to prevent race conditions.

**3. WebSockets for Real-Time Multiplayer Updates**

- Implement real-time notifications for game events using WebSockets to support a multiplayer experience.
- Notify all connected clients of:
    - Round start (multiplier begins increasing).
    - Multiplier updates (at least every 100ms).
    - Player cashouts (including player ID, crypto payout, and USD equivalent).
    - Round crash (including final crash point).
- Allow players to send cashout requests via WebSocket during the round.

**Technical Requirements**

- **Language/Frameworks**:
    - Use Node.js with Express.js.
    - Use NoSQL database (MongoDB) for data storage.
- **WebSocket Library**:
    - Use a library like ws or Socket.IO for Node.js.
- **Crypto API**:
    - Use a free public crypto API like CoinGecko or CoinMarketCap (ensure compliance with rate limits and terms of use).
- **Security**:
    - Validate inputs to prevent invalid bets or cashouts (e.g., negative USD amounts, cashing out after crash).
    - Use a cryptographically secure method for crash point generation.
    - Secure WebSocket messages to prevent abuse (e.g., validate cashout requests).
    - Handle API rate limits and errors gracefully (e.g., fallback to cached prices if the API fails).
- **Documentation**:
    - Provide a README with:
        - Setup instructions, including how to configure the crypto API (e.g., API key if required).

        - API endpoint descriptions (e.g., URL, method, request/response format).
        - WebSocket event descriptions (e.g., event name, payload).
        - Explanation of the provably fair crash algorithm and how it ensures fairness.
        - Details on USD-to-crypto conversion logic and real-time price fetching.
        - Brief overview of your approach to game logic, crypto integration, and WebSockets.

**Deliverables**

1. Source code in a Git repository (e.g., GitHub, GitLab).
2. README with setup and usage instructions, including crypto API setup.
3. A simple script or instructions to populate the database with sample data (e.g., 3-5 player wallets and a few game rounds).
4. A Postman collection or cURL commands to test API endpoints.
5. A basic WebSocket client (e.g., a simple HTML page or script) to demonstrate real-time updates and cashout functionality.

**Evaluation Criteria**

- **Game Logic (35%)**:
    - Correctness of Crash game mechanics and provably fair crash algorithm.
    - Robustness of game state management and round history tracking.
    - Accuracy of multiplier progression and cashout calculations.
- **Cryptocurrency Integration (35%)**:
    - Proper integration of real-time crypto price API and USD-to-crypto conversion.
    - Accurate handling of wallet balances and transaction logging.
    - Atomicity and consistency in balance updates.
- **WebSockets (20%)**:
    - Real-time multiplayer event broadcasting and cashout handling.
    - Scalability and reliability of WebSocket implementation.
- **Code Quality and Documentation (10%)**:
    - Clean, modular, and well-commented code.
    - Comprehensive README and API/WebSocket documentation.

**Submission**

- Submit the Git repository link and any additional files (e.g., Postman collection) via email or a provided platform.
- Ensure the project is deployable locally for testing.

**Notes**

- Focus on backend functionality; no frontend UI is required beyond a basic WebSocket client for testing.(extra points if frontend UI is developed.)
- Simulate cryptocurrency transactions; do not interact with real blockchain networks.
- Ensure the provably fair algorithm is transparent and verifiable.
- Optimize for performance, as Crash games require frequent multiplier updates and real-time responsiveness.
- Handle crypto API rate limits and errors (e.g., cache prices, retry failed requests).
- Ensure the code is production-ready with proper error handling, logging, and concurrency management.

**Important Note**

- Kindly host the backend on render and frontend on netlify/vercel.
- Before sharing a link kindly check if the link is working or not, and it is the correct link, make sure it is publicly available and does not require login.
- If you fail to follow this final procedure your assignment will not get considered.

Good luck, and we look forward to reviewing your submission!