This project is a Cryptocurrency Alert System that allows users to register, set alerts on specific cryptocurrencies, and receive notifications via email when the price of their selected cryptocurrencies crosses a specified target. The system utilizes Node.js, Express, MongoDB, and Socket.IO to handle real-time price updates and alert notifications.

Key Components

1. **User Registration and Authentication:**
   - Users can register with their email, name, number, and address.
   - Passwords are securely hashed using `bcryptjs`.
   - JWT tokens are used for authentication, allowing users to securely access their accounts and set alerts.
2. **Email Notifications:**
   - The system sends email notifications for successful registration, OTP for password reset, and price alerts when a target is reached.
   - Emails are sent using a custom `sendmail` function.
3. **Cryptocurrency Price Tracking:**
   - The application fetches cryptocurrency prices from the CoinGecko API.
   - Prices are cached using `node-cache` to reduce API calls and improve performance.
   - Users can set alerts for specific cryptocurrencies, and the system checks prices periodically to trigger alerts.
4. **Real-time Communication:**
   - Socket.IO is used to establish real-time communication with the client, enabling live updates of cryptocurrency prices.
5. **Alerting System:**
   - Users can set alerts for specific cryptocurrency prices.
   - The system checks prices against user-defined targets every 60 seconds and sends an email alert if the target price is reached.
   - After an alert is triggered, it is removed from the database to avoid repeated notifications.

Challenges Faced and Solutions

1. **Handling Large Volumes of API Requests:**
   - **Challenge:** Fetching cryptocurrency prices frequently from the API could lead to performance issues and API rate limiting.
   - **Solution:** Implemented caching with `node-cache` to store and reuse the latest fetched prices, reducing the frequency of API calls. Additionally, prices are fetched in intervals, allowing efficient use of the API without overwhelming it.
2. **Ensuring Secure User Authentication:**
   - **Challenge:** Ensuring that user credentials are stored and managed securely.
   - **Solution:** Used `bcryptjs` to hash passwords before storing them in the database. JWT tokens are used for secure authentication, allowing users to interact with the system without exposing sensitive data.
3. **Real-time Price Updates:**
   - **Challenge:** Providing real-time updates on cryptocurrency prices to users.

- **Solution:** Utilized Socket.IO to establish a WebSocket connection, allowing the server to push updates to the client as soon as new price data is available. This ensures that users receive the most up-to-date information.

4. **Handling Concurrent Alerts:**
    - **Challenge:** Managing multiple users setting alerts on various cryptocurrencies, which could potentially overwhelm the system.
    - **Solution:** Implemented a systematic check of alerts using a periodic interval. By checking the database every 60 seconds, the system can handle multiple alerts without bottlenecking. Alerts are also deleted after they are triggered, reducing the load on the system.
5. **Email Notification Delivery:**
    - **Challenge:** Ensuring timely and reliable delivery of email notifications for alerts.
    - **Solution:** Utilized a robust email sending function (`sendmail`) that handles different types of email notifications. The system uses try-catch blocks to handle any issues during email sending, ensuring that errors are logged and do not crash the server.
6. **User Data Consistency and Integrity:**
    - **Challenge:** Maintaining consistency in user data, especially during registration, password resets, and alert settings.
    - **Solution:** Used MongoDB with Mongoose to manage user data with proper validation and schema definitions. This ensures that all user actions (like setting alerts or resetting passwords) are consistently reflected in the database.