**Frontend Development with React.js**

**Project Documentation format**

1. **Introduction**
   * **Project Title**: CRYPTPCURRENCY DASHBOARD
   * **Team Members**:
   1. Team Leader: LAKSHMI PRIYA G
   2. Team Member: PRABAVATHI B
   3. Team Member: RAKSHANA R K
   4. Team Member; SHRUTHI C
   5. Team Member: KRITHIKA S K

**PROJECT OVERVIEW**

**Purpose**

A cryptocurrency dashboard is a tool designed to provide users with a comprehensive and real-time overview of the cryptocurrency market. Here's an overview of its purpose

1. Market Monitoring: It allows users to track the performance of various cryptocurrencies, including price changes, market capitalization, and trading volumes.

2. Portfolio Management: Users can manage their cryptocurrency investments by monitoring their holdings, calculating profits or losses, and analyzing trends.

3. Real-Time Updates: The dashboard fetches live data from cryptocurrency exchanges, ensuring users have the latest market information.

4. Decision Support: By offering insights, analytics, and visualizations, it helps users make informed investment decisions.

5. Customization: Many dashboards allow users to personalize their interface, focusing on the metrics and cryptocurrencies most relevant to them.

**Features**

Here are some common features of a cryptocurrency dashboard:

1. Real-Time Market Data

- Display of live prices, market capitalization, trading volumes, and price changes for cryptocurrencies.

- Integration with multiple cryptocurrency exchanges for up-to-date information.

2. Portfolio Management

- Tools to track and manage cryptocurrency holdings.

- Calculation of profits and losses.

- Visual representation of portfolio distribution (e.g., pie charts).

3. Price Alerts

- Customizable notifications for price thresholds.

- Alerts for significant market movements.

4. Historical Data and Charts

- Price trends and historical data visualization.

- Interactive charts for in-depth analysis (e.g., candlestick charts)

5. Analytics and Insights

- Key performance metrics for different cryptocurrencies.

- Sentiment analysis from market trends and news.

- ROI (Return on Investment) calculators.

6. Customizable Interface

- Personalized dashboards tailored to individual preferences.

- Option to focus on specific coins or metrics.

7. News and Updates

- Aggregation of cryptocurrency-related news.

- Updates on regulations, technologies, and market events.

8. Wallet Integration

- Secure connection with cryptocurrency wallets.

- Syncing to track balance and transaction history.

9. Security Features

- Two-factor authentication (2FA) for account safety.

- Data encryption to protect sensitive information.

10. Conversion Tools

- Tools to calculate exchange rates for different cryptocurrencies and fiat currencies.

**ARCHITECHURE**

Architecture of a Cryptocurrency Dashboard

A cryptocurrency dashboard follows a multi-layered architecture to ensure real-time data processing, security, and a smooth user experience. The architecture can be broken down into the following key layers:

1. Frontend Layer (User Interface)

Technologies: HTML, CSS, JavaScript (React, Vue.js, or Angular)

Functionality:

* Displays real-time cryptocurrency prices, charts, and portfolio data.
* Provides interactive graphs, trading features, and alerts.
* Supports user authentication and profile management.

2. Backend Layer (Application Logic & API Handling)

Technologies: Node.js, Python (Django/Flask), or Ruby on Rails

Functionality:

* Handles user authentication and API requests.
* Fetches data from crypto exchanges, blockchain networks, and news sources.
* Processes transactions, stores portfolio data, and triggers alerts.

3. Data Layer (Database & Storage)

Technologies: PostgreSQL, MongoDB, Firebase, or MySQL

Functionality:

* Stores user data, transaction history, and portfolio details.
* Caches frequently accessed data to improve performance.
* Logs activity for security and auditing.

4. External API Integration Layer

APIs Used:

* Crypto market data (Binance, CoinGecko, CoinMarketCap).
* Blockchain data (Etherscan, Blockchain.info).
* News and sentiment analysis (CryptoPanic, Twitter API).

Functionality:

* Fetches live cryptocurrency prices, trading volumes, and market trends.
* Retrieves blockchain transaction details and analytics.
* Gathers news articles and social media trends for sentiment analysis.

5. Security Layer

Technologies: OAuth, JWT, SSL/TLS encryption, 2FA

Functionality:

* User authentication and role-based access control.
* Secure API communication with encryption.
* Fraud detection and anti-phishing measures.

6. Notification & Alert System

Technologies: Firebase Cloud Messaging (FCM), Twilio, WebSockets

Functionality:

* Sends price alerts, transaction confirmations, and system notifications.
* Provides push notifications, email, or SMS updates for important events.

7. Deployment & Hosting Layer

Technologies: AWS, Google Cloud, Azure, Docker, Kubernetes

Functionality:

* Scales the dashboard based on user demand.
* Ensures high availability and low-latency performance.
* Automates deployment and updates with CI/CD pipelines.

**SETUP INSTRUCTIONS**

**Prerequisites**

Ensure the following software and tools are installed on your system before proceeding:

Node.js (version 14.x or higher)

npm (Node Package Manager, bundled with Node.js) or Yarn for package management

Git for version control

A modern web browser (e.g., Google Chrome, Mozilla Firefox)

A text editor or IDE, such as Visual Studio Code

An API key from a cryptocurrency data provider like CoinGecko or CoinMarketCap

Installation Steps

1. Clone the Repository

Clone the project repository to your local machine using Git:

git clone <repository-url>

Replace `<repository-url>` with the repository's actual URL.

1. Navigate to the Project Directory

Move into the project folder with this command:

cd <project-folder>

Replace `<project-folder>` with the folder name where the repository was cloned.

3. Install Project Dependencies

Install all required dependencies by running:

npm install

Alternatively, use Yarn:

yarn install

4. Set Up Environment Variables

- Create a `.env` file in the root directory of the project.

- Add the following environment variables to the `.env` file:

env

REACT\_APP\_API\_KEY=<your-api-key>

REACT\_APP\_API\_URL=https://api.cryptodata.com

Replace `<your-api-key>` with the API key you received from the cryptocurrency data provider.

5. Start the Development Server

Launch the application by running the following command:

npm start

If you're using Yarn, execute:

yarn start

6. Access the Dashboard

Open your preferred web browser and navigate to:

http://localhost:3000

The cryptocurrency dashboard will now be accessible locally.

**FOLDER STURCTURE**

**Client**

The project follows a modular structure to ensure scalability:

/client

│── /src

│ │── /components # Reusable UI components (cards, charts, tables)

│ │── /pages # Page-level components (Dashboard, Wallet, Settings)

│ │── /assets # Static assets (logos, icons, images)

│ │── /hooks # Custom React hooks (useFetchCrypto, useTheme)

│ │── /utils # Helper functions (API requests, formatting)

│ │── /context # Context API for global state (theme, authentication)

│ │── /services # API services for cryptocurrency data fetching

│ │── App.js # Main application component

│ │── index.js # Entry point of the application

│── package.json # Dependencies and scripts

│── .env # Environment variables (API keys, endpoints)

│── README.md # Project documentation

**Utilities**

* **Helper Functions (/utils)**:
  + formatCurrency.js – Formats cryptocurrency values.
  + calculateChange.js – Computes percentage price changes.
  + api.js – Handles API requests to fetch live cryptocurrency data.
* **Custom Hooks (/hooks)**:
  + useFetchCrypto.js – Fetches live cryptocurrency prices.
  + useTheme.js – Manages theme switching between dark and light modes.
* **Context API (/context)**:
  + AuthContext.js – Manages authentication and user session.
  + ThemeContext.js – Handles theme preferences.

**RUNNING APPLICATION**

**Frontend**

To start the cryptocurrency dashboard locally, run:

cd client

npm install # Install dependencies

npm start # Start the development server

This runs the application on http://localhost:3000/ by default.

**COMPONENT DOCUMENTATION**

**Key Components**

* **CryptoTable** (CryptoTable.js) – Displays a list of cryptocurrencies with real-time prices.
* **PriceChart** (PriceChart.js) – Interactive chart for tracking price trends.
* **PortfolioOverview** (PortfolioOverview.js) – Shows the user's cryptocurrency holdings and performance.

**Reusable Components**

* **Card** (Card.js) – A generic card component used across the dashboard.
* **Button** (Button.js) – Customizable button with props like variant, size, and onClick.
* **Modal** (Modal.js) – Generic modal for popups and confirmations.

**STATE MANAGEMENT**

**Global State**

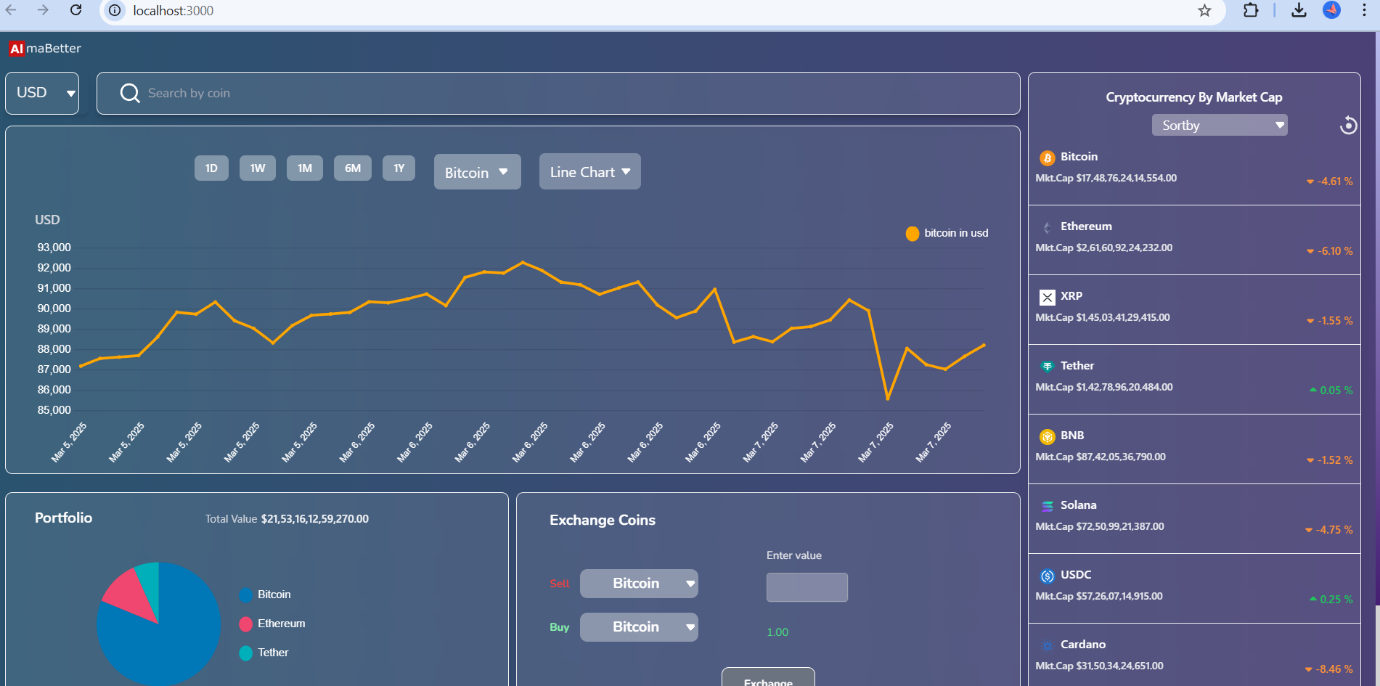
* Managed using **React Context API** or **Redux**.
* Stores cryptocurrency data, user authentication, and theme preferences.

**Local State**

* Managed using **React's useState hook**.
* Used for handling UI interactions like search filters, toggles, and form inputs.

**USER INTERFACE**

This is the user interface for the cryptocurrency dashboard



**STYLING**

**CSS Frameworks/Libraries**

* Used for responsive layouts, pre-designed UI components, and faster development.
* **Sass/SCSS** (if applicable) – Enables the use of variables, mixins, and nesting to streamline CSS management.
* **CSS-in-JS (e.g., Styled-Components, Emotion)** – If applicable, mention its usage in handling dynamic styling within JavaScript frameworks.

**Theming & Custom Design Systems**

* **CSS Variables (--primary-color)** – Used for dynamic theme switching and maintaining a unified color scheme.
* **Preprocessor (Sass/SCSS) Variables** – Helps in managing colors, fonts, and spacing centrally.
* **Custom Design System** – Defines a reusable set of UI components, typography, and color palettes to maintain brand identity.
* **Dark/Light Mode Support** – Implemented using CSS variables, JavaScript, or framework-specific theming capabilities.

This theming approach ensures a consistent UI experience while allowing flexibility for customization.

**TESTING**

**Testing Strategy**

1. Unit Testing

- Objective: Verify the smallest components of the application (e.g., API handlers, functions, and UI elements).

- Tools: Jest, Mocha, or Pytest.

- Example: Test API functions for fetching live cryptocurrency prices and their accurate parsing.

2. Integration Testing

- Objective: Test interactions between different modules, such as APIs and the user interface.

- Tools: Cypress or Selenium.

- Example: Ensure that changes in the backend API accurately reflect on the front end (e.g., updating a cryptocurrency’s market price

3. System Testing

- Objective: Validate the entire system's functionality as a whole.

- Scope:

- Live data updates.

- Portfolio management.

- Navigation across pages.

- Tools: Manual testing and tools like TestRail for tracking.

4. Regression Testing

- Objective: Ensure updates or fixes don’t break existing functionality.

- Tools: Automated scripts using tools like Selenium or Puppeteer.

5. Performance Testing

- Objective: Assess the dashboard’s performance under varying loads.

- Metrics:

- Response time for API calls.

- Memory usage during real-time updates.

- Tools: JMeter or Lighthouse.

6. Security Testing

- Objective: Identify and mitigate vulnerabilities, such as unauthorized access or insecure API endpoints.

- Tools: OWASP ZAP or Burp Suite.

7. Usability Testing

- Objective: Ensure that the dashboard provides an intuitive and user-friendly experience.

- Approach: Conduct user surveys and feedback sessions.

**Code Coverage**

Definition: Code coverage measures the percentage of your application’s source code that is executed during testing, ensuring all functionalities are adequately tested.

1. Coverage Metrics

- Function Coverage: Ensure all functions are executed at least once.

- Branch Coverage: Test all possible code branches (e.g., `if`, `else` conditions).

- Statement Coverage: Confirm all lines of code are executed.

- Path Coverage: Validate all possible execution paths in your code.

2. Tools for Measuring Coverage

- JavaScript: Istanbul/NYC or Jest with built-in coverage reports.

- Python: Coverage.py.

- Java: JaCoCo or Cobertura.

3. Example Coverage Goals

- Aim for 80-90% coverage in critical modules like API interactions and portfolio calculations.

- Identify and prioritize areas with less than 60% coverage for improvements.

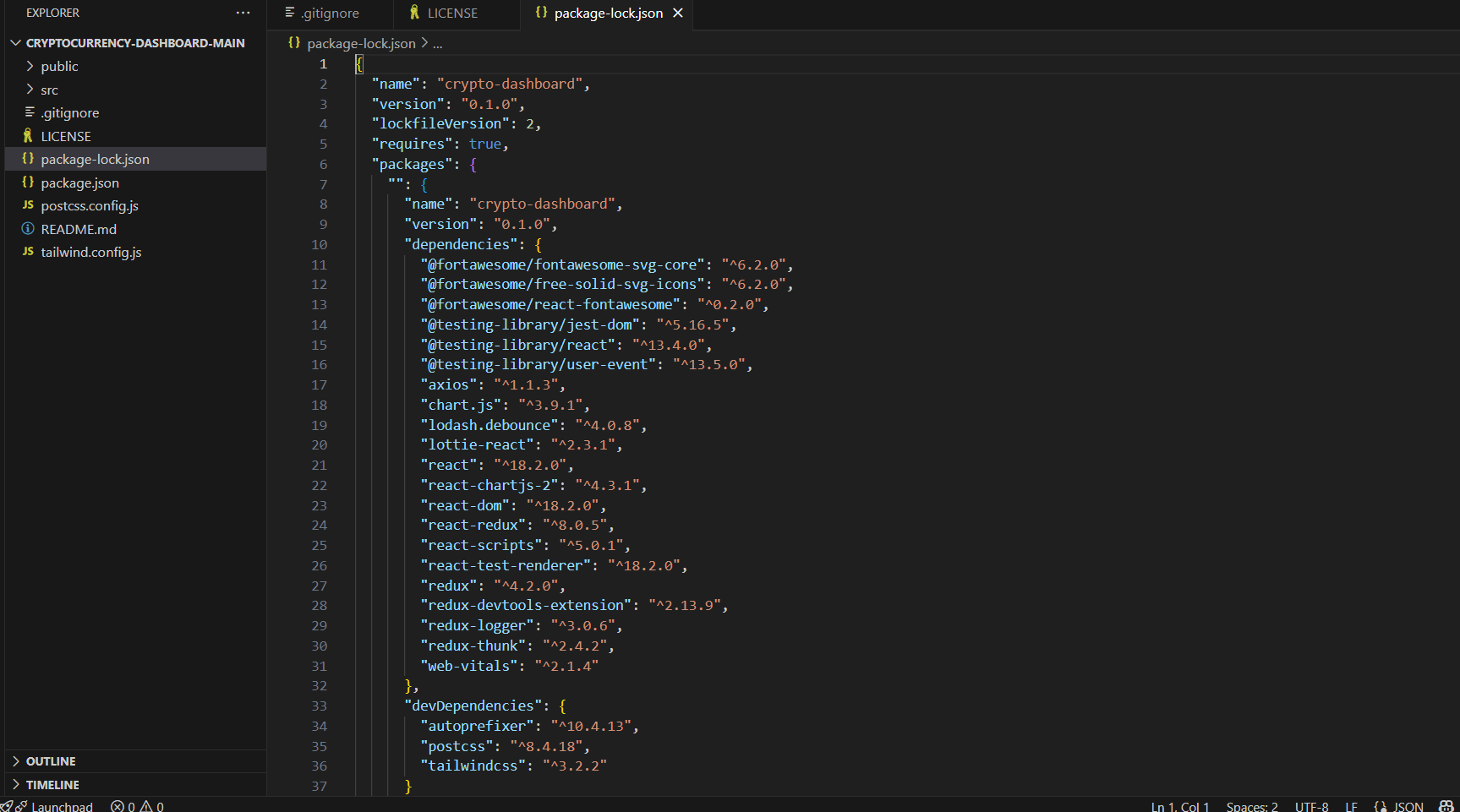
4. Reporting

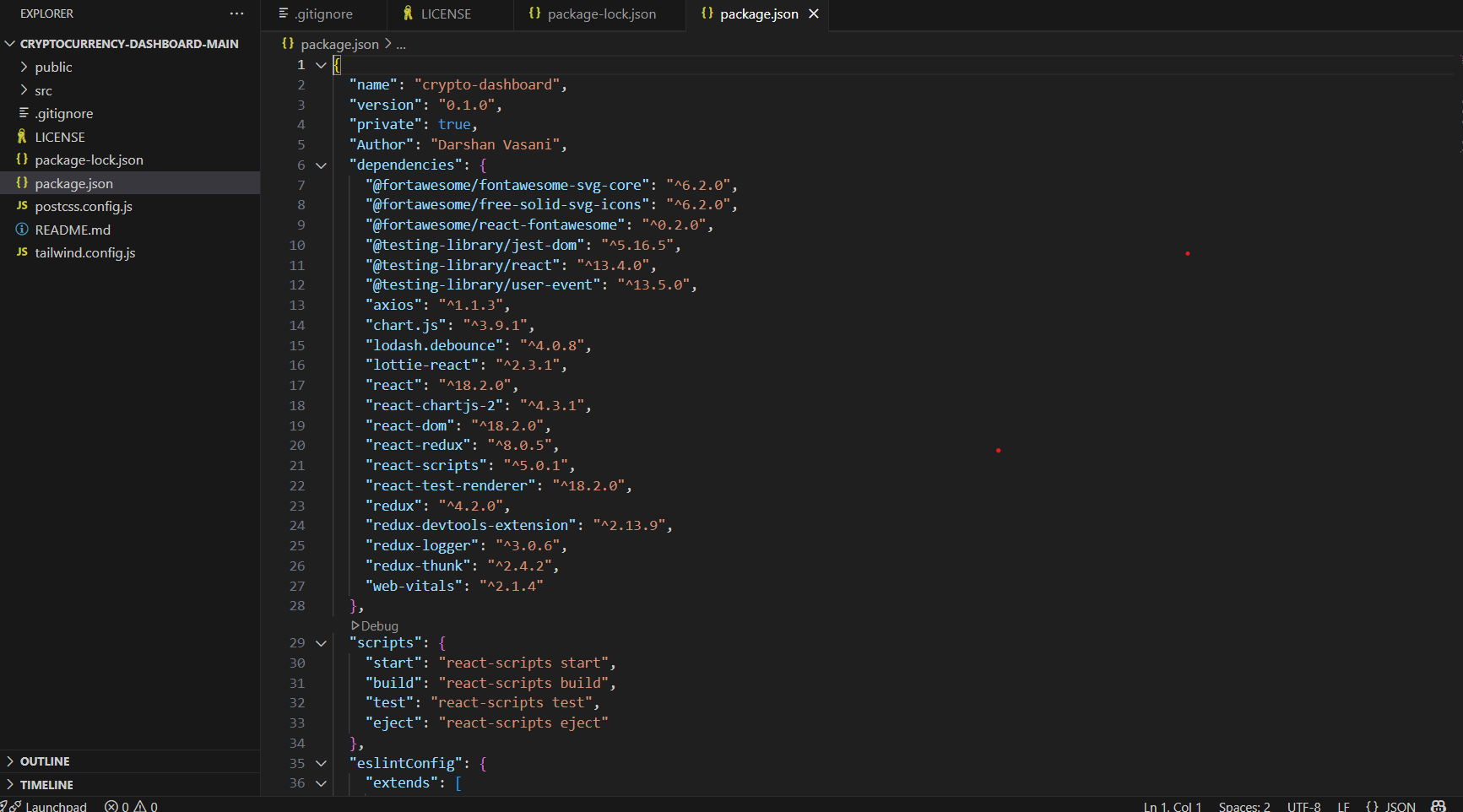
- Generate coverage reports using tools like Jest or Coverage.py.

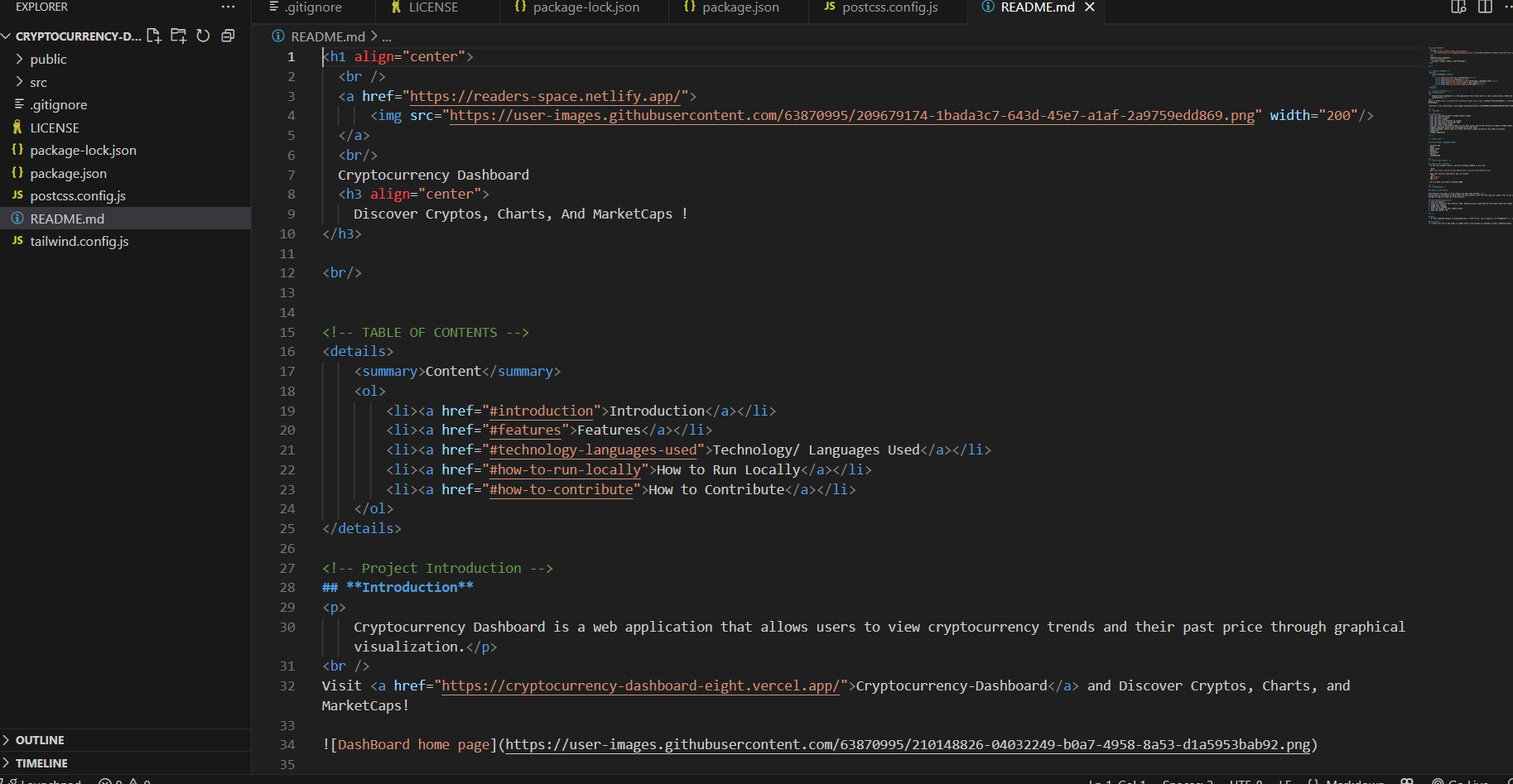
- Visualize reports with HTML output for better understanding and debugging.

**SCREENSHOTS OR DEMO**

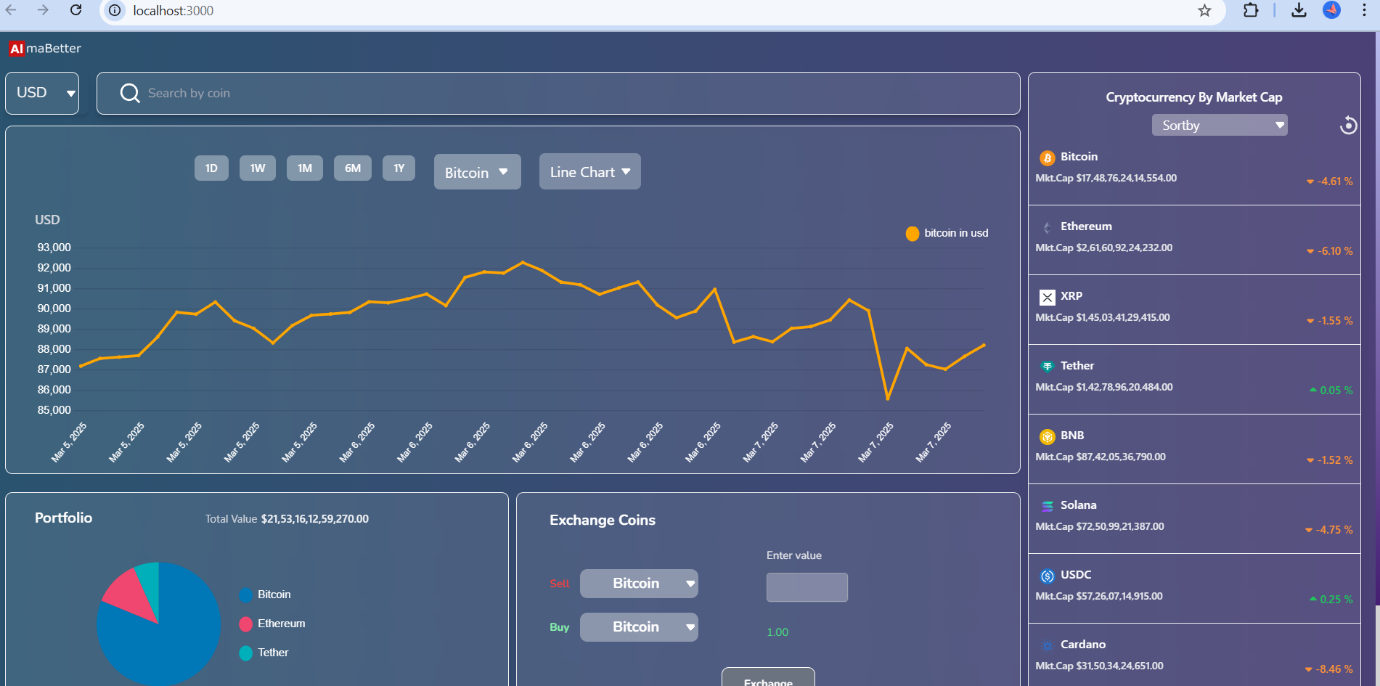
**CODING**

****

****

****

**OUTPUT SCREENSHORT**





DEMO LINK

<https://drive.google.com/file/d/1qs25_MrtRyeKE6y1hJh1mYO6PL63g7tl/view?usp=sharing>

**KNOWN ISSUES**

The following are known bugs or issues that may affect the cryptocurrency dashboard:

| **Issue** | **Description** | **Workaround (if any)** |
| --- | --- | --- |
| **Delayed API Updates** | Cryptocurrency prices may not refresh instantly due to API rate limits. | Implement WebSockets or increase polling frequency. |
| **Chart Data Lag** | The price chart may not update immediately when switching timeframes. | Optimize state management and ensure proper API caching. |
| **Dark Mode Glitches** | Some UI elements may not switch colors properly when toggling between light and dark modes. | Review theme context and CSS variables. |
| **Mobile Responsiveness** | Some table layouts may break on smaller screens. | Improve CSS media queries for better responsiveness. |

**FUTURE ENHANCEMENT**

**Planned Features & Improvements**

* **WebSocket Integration**: Implement real-time price updates instead of periodic API polling.
* **Advanced Portfolio Management**: Add features like transaction history, profit/loss tracking, and performance analysis.
* **Customizable Dashboards**: Allow users to personalize the layout, add/remove widgets, and set alerts.
* **Animations & Microinteractions**: Improve UI engagement with smooth animations for transitions and hover effects.
* **Multi-Language Support**: Add localization for different languages to enhance accessibility.
* **Dark Mode Improvements**: Ensure full compatibility across all UI components.
* **Mobile Optimization**: Enhance the mobile experience by refining layouts for smaller screens.