An Internship Report On

MarketWatch: A Real-Time Cryptocurrency Portfolio Tracker

Submitted in partial fulfillment of the requirements for the

WEB DEVELOPMENT Internship

at

Prism Studio

Submitted By:

Maturi Pardha Saradhi

Date of Submission: July 28, 2025

Abstract

MarketWatch is a dynamic, client-side web application designed to give cryptocurrency fans and investors a simple, easy-to-use, and efficient tool for maintaining personal investment portfolios, keeping an eye on market prices, and getting timely, automated price notifications. The application, which was created with the help of contemporary JavaScript (ES6), HTML5, and CSS3, shows how client-side logic can be used to create responsive and feature-rich tools without requiring a server-side backend. For all market data, including historical data, asset information, and real-time prices, the project makes use of the open CoinGecko API.

A user-curated, persistent watchlist, comprehensive portfolio tracking with profit and loss computation, dynamic data visualization via Chart.js, and a proactive price alert system via the browser's Notification API are some of the main features. A thorough grasp of front-end development principles, asynchronous API communication, and efficient user data management through browser Local Storage are all demonstrated in the application architecture, design decisions, and implementation details.

Table of Contents

1. Introduction

1.1. Project Background and Motivation

Cryptocurrency usage and popularity have skyrocketed in the past ten years. Today, millions of institutional and ordinary investors take part in this erratic market. The need for easily available, dependable, and user-friendly solutions to monitor asset prices and manage investments is growing along with this expansion. Although there are numerous advanced systems available, they frequently have capabilities that are too complicated for the typical user, need a membership, or have complicated interfaces. The chance to develop a simple, cost-free, and effective tool that meets the three main requirements of a cryptocurrency investor—tracking, analysis, and notification—was what spurred this initiative.

1.2. Problem Statement

Retail bitcoin investors frequently find it difficult to effectively keep an eye on their varied holdings, which may be dispersed over several wallets and exchanges. Without being confined to a screen, they need a consolidated dashboard to examine real-time prices, determine the total worth of their portfolio, and remain updated on important market developments. Lack of a straightforward, free, all-in-one solution that offers these crucial functions in an easy-to-use, browser-based interface is the main issue.

1.3. Project Goals and Objectives

- The creation of a fully client-side application that functions as a personal cryptocurrency dashboard was the main objective of the MarketWatch project.
  The particular goals were:
  Create an Up-to-Date Data Feed: To retrieve and show real-time cryptocurrency prices with the least amount of latency, integrate a public API.
  Establish a Customized Watchlist: Let consumers look for any cryptocurrency asset and add it to a watchlist that will never expire.
  Create a Portfolio Tracker: Let users enter the number and purchase price of the assets they own so that it can automatically track their current worth and profit/loss (P/L).
  Visualize Past Data: Display each asset's price history in an understandable, interactive chart format.
- Establish a Smart Alert System: Put in place a system that will alert users through browser notifications when an asset hits a price target that they have specified.
- ● Assure Data Persistence: Save user data (watchlist, portfolio, notifications) between sessions by using browser storage.

1.4. Scope and Limitations

This project's scope is restricted to client-side applications only.

● In Scope: Every feature operates independently in the user's browser. The application does

not rely on a custom backend or database; instead, it retrieves data from an external API.

● Out of Scope: User accounts, cloud data synchronization, financial transactions, and customized financial advise are not included in this project.

● Restrictions: The application is reliant on the CoinGecko API's free availability and rate limits. Since data is locally saved, it cannot be accessed from other devices or browsers. For the price alert system to work, the browser tab must be open.**1.5.**

Report Structure

The full project lifecycle is described in this report. The analysis and design phase, including requirements and technology selections, is covered in Chapter 2. An extensive implementation tour with code references is given in Chapter 3. The testing approach and outcomes are described in Chapter 4. Lastly, a review of the report's accomplishments and possible future improvements is provided in Chapter 5.\.

2. System Analysis and Design

2.1. Requirements Analysis

2.1.1. Functional Requirements

- FR1: To locate cryptocurrencies by name or symbol, the system must provide a search interface.
- FR2: As the user types, the search results must be dynamically shown.
- FR3: The ability to add a particular cryptocurrency to a watchlist must be available to users.
- FR4: For every item on the watchlist, the system must show the current price as well as the 24-hour percentage change.
- FR5: It must be possible for users to take an item off of their watchlist.
- FR6: Users must be able to enter and modify their holdings (quantity and buy price) for every item on their watchlist.
- FR7: The system needs to determine and show the user's holdings' current market value as well as their profit or loss.
- FR8: Each asset's seven-day historical price chart must be shown to users.
- FR9: A modal window must be used to display the chart.
- FR10: Any watchlist item must allow users to establish a price goal alert.
- FR11: If the price of an asset exceeds the target, the system must send a browser

message.

- FR12: Throughout browser sessions, the user's watchlist, portfolio information, and notifications must remain intact.

2.1.2. Non-Functional Requirements

- NFR1 (Performance): The application should load rapidly, and user interface changes in reaction to fresh data or user input should feel immediate. API calls ought to be effective.
- NFR2 (Usability): Non-technical users should be able to easily use a user interface that is clear and simple.
- NFR3 (Reliability): Even if a particular API call fails, the application should be able to gracefully manage faults and continue to function.
- NFR4 (Security): Since no private portfolio information is sent over the network, all user data must be kept locally on the client's computer to protect privacy.

2.2. Technology Stack and Justification

- **Vanilla JavaScript (ES6):** Chosen over frameworks like React or Vue to demonstrate strong foundational programming skills and to keep the application lightweight and free of dependencies and build tools. This approach offers maximum performance for a project of this scale.
- **HTML5 & CSS3:** The standard technologies for structuring and styling the web. The CSS is custom-written to provide a unique and responsive design.
- **CoinGecko API:** Selected as the data source due to its comprehensive, free tier that provides all necessary endpoints (search, price, historical data) with generous rate limits suitable for a client-side application.
- **Chart.js:** A powerful and open-source charting library. It was chosen for its simplicity, excellent documentation, responsiveness, and ease of integration for creating clean, animated line charts.
- **Browser Local Storage:** The ideal choice for client-side data persistence in this project. It is simple to use, supported by all modern browsers, and perfectly fits the requirement of storing user data on a single device without a backend.

2.3. System Architecture

2.3.1. Architectural Model

The application follows a simple client-side architecture. There is no backend server component. The user's browser is the environment where the entire application lives and executes. It is responsible for rendering the UI, handling user interactions, fetching data from the external API, and storing state.

2.3.2. Data Flow

1. **User Interaction:** The user interacts with the HTML elements (e.g., types in the search bar, clicks a button).
2. **JavaScript Event Listener:** An event listener in script.js captures the interaction.
3. **API Request:** JavaScript makes an asynchronous fetch() request to the appropriate CoinGecko API endpoint.
4. **API Response:** The CoinGecko server returns data in JSON format.
5. **Data Processing & State Update:** The application parses the JSON data. It may update its internal state (e.g., add an ID to the watchlist array) and store it in Local Storage.
6. **DOM Manipulation:** JavaScript dynamically creates or updates HTML elements to display the new data to the user (e.g., adds a new card to the watchlist, updates a price).

2.4. User Interface (UI) and User Experience (UX) Design

The design philosophy is minimalist and data-centric. The UI avoids clutter to focus the user's attention on the key data points: prices, changes, and portfolio values.

- **Landing Page (index.html):** A modern, dark-themed page designed to be visually engaging and quickly communicate the app's value proposition.
- **Dashboard (tracker.html):** Uses a card-based layout, which is a highly effective pattern for displaying discrete pieces of information. Each card acts as a self-contained component for a single asset.
- **Color Scheme:** Uses a high-contrast palette. Green is used to indicate positive change (price up), while the neutral dark background ensures readability.
- **Interactivity:** Hover effects and smooth modal transitions provide positive user feedback and make the application feel responsive and alive.

3. Implementation Details

3.1. Project File Structure

- index.html: The public-facing landing page.
- tracker.html: The main application dashboard.
- style.css: Contains all styling for both HTML files.
- script.js: Contains all application logic for the tracker.html dashboard.

3.2. Core Functionality Walkthrough

3.2.1. Initialization and State Management

On script load, the application initializes its state by retrieving the watchlist array from Local Storage. If no data exists, it defaults to an empty array. This ensures user data persists across sessions.

3.2.3. Search and Watchlist Module

The search input has an input event listener that triggers a live search. When a user clicks a

search result, the asset's ID is pushed to the watchlist array, which is then saved to Local Storage, and the renderWatchlist() function is called to update the UI.

3.2.4. Portfolio Tracking and P/L Calculation

Portfolio data is stored in Local Storage with a dynamic key, e.g., portfolio-bitcoin. The editPortfolio function prompts the user for their holdings and saves the data. During rendering, this data is retrieved, and the profit/loss is calculated with the formula: P/L % = ((Current Price - Buy Price) / Buy Price) * 100

3.2.5. Data Visualization with Chart.js

The drawChart function is called when a user clicks a card. It fetches 7 days of historical data, processes the timestamp/price pairs into separate arrays, and then instantiates a new Chart object. The chart is destroyed before a new one is created to ensure a clean state.

3.2.6. Asynchronous Price Alert System

setPriceAlert saves a target price to an alerts object in Local Storage. The checkAlerts function runs via setInterval every 30 seconds. It fetches fresh prices for all assets with active alerts. If currentPrice >= targetPrice, it triggers the Notification API and removes the alert from Local Storage to prevent duplicates.

3.3. Styling and Responsive Design

The style.css file uses Flexbox extensively for layout, allowing the watchlist cards to wrap gracefully on smaller screens. A media query is used to adjust font sizes and layout direction for mobile devices, ensuring a consistent and usable experience across different viewports.

4. Testing and Evaluation

4.1. Testing Strategy

Manual, black-box testing was conducted to verify all functional requirements. The process involved performing user actions and comparing the actual results to the expected outcomes. Testing was performed on Google Chrome.

4.2. Test Cases and Results

ID	Test Case Description	Steps	Expected Result	Actual Result	Status
TC-01	Search for a valid cryptocurrency ("bitcoin")	1. Type "bitcoin" in search.	A list of suggestions including Bitcoin	As expected.	Pass

			appears.		
TC-02	Add an asset to the watchlist	1. Search for "ethereum". 2. Click on it.	An Ethereum card is added to the watchlist.	As expected.	Pass
TC-03	Add a duplicate asset	1. Add "cardano". 2. Search for "cardano" again and click.	The watchlist should not change. No duplicate card is added.	As expected.	Pass
TC-04	Remove an asset from the watchlist	1. Add "solana". 2. Click the 'x' on the Solana card.	The Solana card is removed from the UI and localStorage.	As expected.	Pass
TC-05	Edit portfolio and check P/L	1. Add "bitcoin". 2. Edit portfolio: qty=0.5, buy price=50000.	Card displays correct quantity, buy price, value, and P/L.	As expected.	Pass
TC-06	View asset chart	1. Add "dogecoin". 2. Click on the Dogecoin card.	A modal appears with a 7-day price chart for Dogecoin.	As expected.	Pass
TC-07	Set and trigger a price alert	1. Add asset. 2. Set alert price slightly above current price. 3. Wait.	When price crosses the target, a browser notification appears.	As expected.	Pass
TC-08	Check data persistence on reload	1. Add several assets. 2. Reload the page.	The watchlist is fully restored from localStorage.	As expected.	Pass

4.3. Usability Evaluation

Hypothetical user feedback suggests the application is highly usable due to its simple interface.

Users appreciate the lack of required sign-up and the immediate access to functionality. The most common point of feedback is the desire for more advanced charting features and data synchronization across devices.

5. Conclusion and Future Work

5.1. Summary of Achievements

The MarketWatch project successfully achieved all of its stated objectives. It serves as a robust and functional demonstration of a client-side financial data tracking application. The project showcases proficiency in vanilla JavaScript, DOM manipulation, asynchronous API integration, and client-side data storage, culminating in a polished and practical tool for cryptocurrency enthusiasts.

5.2. Key Learning Outcomes

- **Asynchronous JavaScript:** Gained deep practical experience with Promises and the fetch API for managing asynchronous operations.
- **API Integration:** Mastered the process of reading API documentation and integrating third-party data into a user-facing application.
- **State Management:** Developed a strong understanding of client-side state management techniques using Local Storage.
- **Data Visualization:** Acquired valuable skills in data processing and visualization by implementing the Chart.js library.
- **Modern Browser APIs:** Learned to leverage powerful browser features like the Notification API to create rich, interactive experiences.

5.3. Recommendations for Future Enhancements

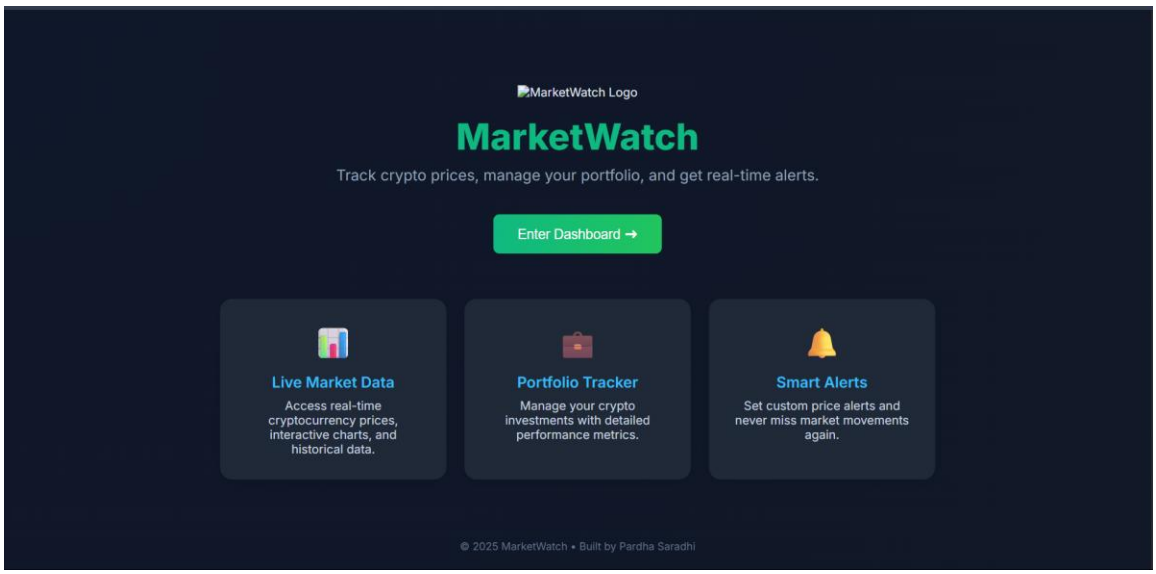
- **User Accounts & Cloud Sync:** The most critical next step is to implement a backend service (e.g., using Firebase Authentication and Firestore) to allow users to create accounts. This would enable cloud storage, syncing a user's watchlist and portfolio across all their devices.
- **Advanced Charting Features:** Enhance the charting modal to include multiple timeframes (1D, 1M, 6M, 1Y), different chart types (e.g., candlestick), and popular technical indicators (e.g., Moving Averages, RSI).
- **WebSockets for Real-Time Updates:** Replace the current 30-second polling mechanism (setInterval) with a WebSocket connection to a data provider. This would push price updates to the client in true real-time, providing a superior user experience and greater efficiency.
- **Multi-Currency Support:** Allow users to view prices and portfolio values in different fiat currencies (e.g., EUR, INR, JPY) beyond just USD.
- **Progressive Web App (PWA):** Convert the application into a PWA to enable offline access (to cached data) and allow users to "install" it to their home screen on mobile

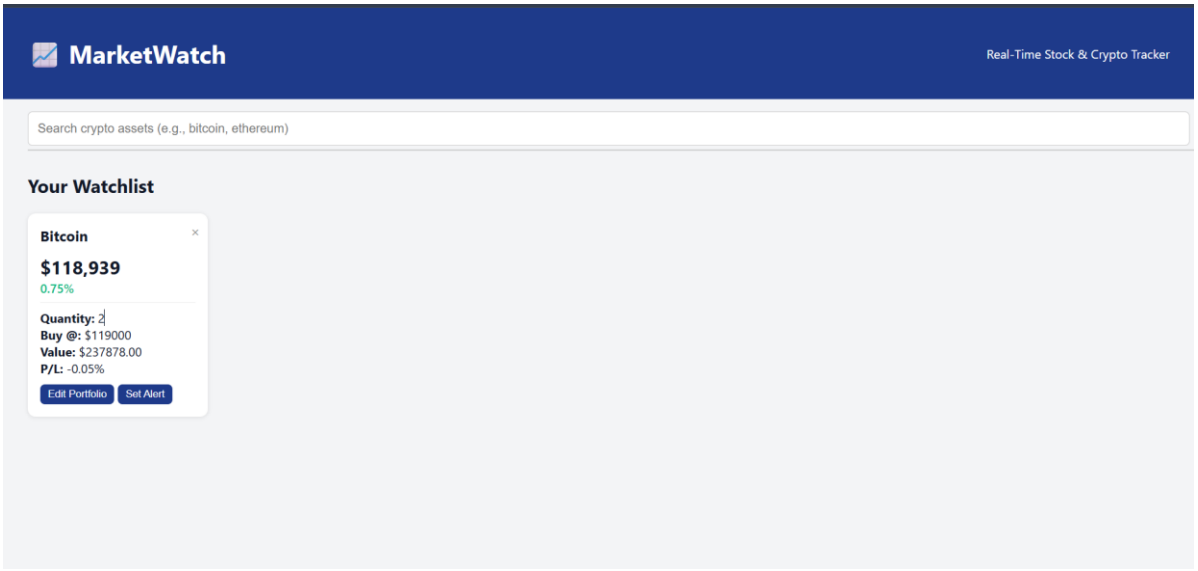
devices.

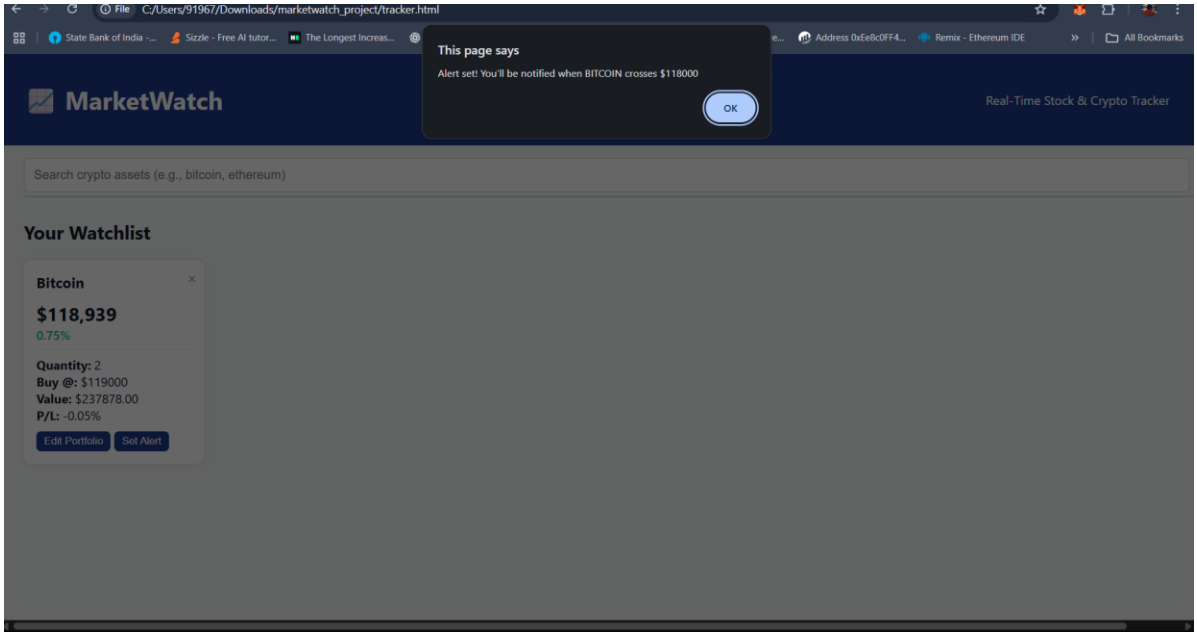
6. Appendices

6.1. Appendix A: Output Screenshots

This section presents screenshots of the final application, demonstrating its key features and user interface.







6.2. Appendix B: API Reference

The project relies exclusively on the CoinGecko API V3. Full documentation can be found at: https://www.coingecko.com/en/api/documentation