

Crafting Crypto Tracker Landing Page

Session Overview

In this session, we will start by designing the **layout** of our Crypto Tracker and then move on to fetching **real-time crypto data** from an API. By the end of the class, you'll have a working landing page that displays live cryptocurrency prices.

By the end of this session, you will:

- Build a **responsive cryptocurrency listing page**.
- Use **CSS Grid and Flexbox** to structure and style web layouts.
- Fetch **real-time cryptocurrency data** using the Fetch API.
- Handle **API errors and optimize performance**.

Creating a Responsive Crypto Tracker Landing Page

Layout for the landing page:

For creating essential files, start with setting up the HTML document, and understand the importance of semantic elements in building a well-organized and accessible web page for Crypto Tracker as follows,

```
/crypto-tracker
├── index.html      # Main HTML file
├── styles.css      # CSS for styling the page
└── app.js         # JavaScript file to fetch API data
```

HTML File (Structural layout):

The **index.html** file serves as the backbone of the project. It defines the web page's structure using **semantic HTML elements** for better accessibility and SEO.

A basic HTML boilerplate is to be initiated first to structure the elemental layout of our crypto tracker,

Index.html File(Reference)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Metadata and External Stylesheets -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cryptocurrency Dashboard</title>

  <!-- Linking Custom Stylesheet -->
  <link rel="stylesheet" href="style.css">

  <!-- Linking Font Awesome for Icons -->
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css"
>
</head>

<body>
  <!-- Navigation Bar -->
  <nav>
    <div class="left-nav">
      <!-- Website Title and Home Link -->
      <a href="index.html" class="title">CRYPTO TRACKER</a>
    </div>
    <div class="right-nav">
      <!-- Navigation Links -->
      <a href="index.html" class="nav-link">Home</a>
      <a href="favorites.html" class="nav-link">Liked Coins</a>

      <!-- Search Bar with Icon -->
      <div class="search-container">
        <input type="text" id="search-box" placeholder="Search Coins..."
autocomplete="off">
        <i class="fas fa-search" id="search-icon"></i>

        <!-- Search Results Dialog Box -->
        <div class="dialog-box" id="search-dialog">
```

```
<div class="dialog-content">
  <!-- Close Button for Dialog Box -->
  <span id="close-dialog" class="close-btn">&times;</span>
  <h2>Search Results</h2>
  <ul id="search-results">
    <!-- Search results will be dynamically inserted by
JavaScript -->
  </ul>
</div>
</div>
</div>
</div>
</nav>

<!-- Main Content -->
<main>
  <!-- Table for Displaying Cryptocurrency Data -->
  <table id="crypto-table">
    <thead>
      <tr>
        <th>#</th>
        <th>Icon</th>
        <th>Name</th>
        <th>Price
          <!-- Sorting Icons for Price Column -->
          <i id="sort-price-asc" class="fas fa-arrow-up"></i>
          <i id="sort-price-desc" class="fas fa-arrow-down"></i>
        </th>
        <th>Volume
          <!-- Sorting Icons for Volume Column -->
          <i id="sort-volume-asc" class="fas fa-arrow-up"></i>
          <i id="sort-volume-desc" class="fas fa-arrow-down"></i>
        </th>
        <th>Market Cap
          <!-- Sorting Icons for Market Cap Column -->
          <i id="sort-market-asc" class="fas fa-arrow-up sort-icon"
title="Sort by market ascending"></i>
          <i id="sort-market-desc" class="fas fa-arrow-down
sort-icon" title="Sort by market descending"></i>
        </th>
      </tr>
    </thead>
  </table>
</main>
</div>
```

```
        </th>
        <th>Favorite</th>
    </tr>
</thead>
<tbody>
    <!-- Cryptocurrency data will be dynamically inserted here using
JavaScript -->
</tbody>
</table>

    <!-- Placeholder for Pagination and Loading Animation -->
</main>

<!-- JavaScript File -->
<script src="script.js"></script>
</body>
</html>
```

Representation:

CRYPTO TRACKER

Home Liked Coins Search Coins...

Icon Name Price ↑ Volume ↑ Market Cap ↑ Favorite

Explanation:

This `index.html` file forms the **core structure** of a **cryptocurrency tracking dashboard**. It includes:

1. **Head Section (<head>)** – Defines metadata, links CSS, and includes Font Awesome icons.
2. **Navigation Bar (<nav>)** – Provides links to the homepage and liked coins.

3. **Search Box** (`<div class="search-container">`) – Allows users to search for cryptocurrencies dynamically.
4. **Cryptocurrency Table** (`<table id="crypto-table">`) – Displays real-time crypto data in a structured format.
5. **Sorting Buttons** – Enables price, volume, and market cap sorting.
6. **JavaScript Integration** (`script.js`) – Fetches API data, updates the table, and handles interactions.

Components of HTML File:

1. Head Section (`<head>`)

Code →

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cryptocurrency Dashboard</title>
  <link rel="stylesheet" href="style.css">

  <!-- Used to include an external CSS stylesheet from Font Awesome, a popular icon
  library-->
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css"
>
</head>
```

Explanation:

- Defines the document's metadata.
- Ensures mobile responsiveness (`meta viewport`).
- Links **CSS style sheets** for custom styling (`style.css`).
- Imports **Font Awesome** icons for UI elements like search and sorting buttons. (An external CSS stylesheet)

2. Navigation Bar (<nav>)

Code →

```
<nav>
  <div class="left-nav">
    <a href="index.html" class="title">CRYPTO TRACKER</a>
  </div>
  <div class="right-nav">
    <a href="index.html" class="nav-link">Home</a>
    <a href="favorites.html" class="nav-link">Liked Coins</a>
  </div>
</nav>
```

Explanation:

- Provides **easy navigation** between the homepage and the favorites page.
- The **left section (left-nav)** displays the website title.
- The **right section (right-nav)** contains:
 - A **Home link (index.html)**.
 - A **Favorites link (favorites.html)** for tracking liked cryptocurrencies.

3. Search Box (<div class="search-container">)

Code →

```
<div class="search-container">
  <input type="text" id="search-box" placeholder="Search Coins..."
autocomplete="off">
  <i class="fas fa-search" id="search-icon"></i>
  <div class="dialog-box" id="search-dialog">
    <div class="dialog-content">
      <span id="close-dialog" class="close-btn">&times;</span>
      <h2>Search Results</h2>
      <ul id="search-results">
        <!-- Search results will be populated here -->
      </ul>
    </div>
  </div>
</div>
```

Explanation:

- Provides a **search functionality** for cryptocurrencies.
- Users **type in the input box** (`<input id="search-box">`).
- A **search icon** (`<i class="fas fa-search">`) is displayed for better UI.
- A **popup dialog** (`<div id="search-dialog">`) appears, showing results dynamically fetched from an API.

4. Cryptocurrency Table (`<table id="crypto-table">`)

Code →

```
<main>
  <table id="crypto-table">
    <thead>
      <tr>
        <th>#</th>
        <th>Icon</th>
        <th>Name</th>
        <th>Price
          <i id="sort-price-asc" class="fas fa-arrow-up"></i>
          <i id="sort-price-desc" class="fas fa-arrow-down"></i>
        </th>
        <th>Volume
          <i id="sort-volume-asc" class="fas fa-arrow-up"></i>
          <i id="sort-volume-desc" class="fas fa-arrow-down"></i>
        </th>
        <th>Market Cap
          <i id="sort-market-asc" class="fas fa-arrow-up
sort-icon"></i>
          <i id="sort-market-desc" class="fas fa-arrow-down
sort-icon"></i>
        </th>
        <th>Favorite</th>
      </tr>
    </thead>
    <tbody>
      <!-- Table content will be inserted here by JavaScript -->
    </tbody>
  </table>
</main>
```

Explanation:

- The **main content area** where **real-time cryptocurrency data** is displayed.
- The `<table>` element consists of:
 - **Table header (`<thead>`):**
 - **#** (Serial number).
 - **Icon** (Cryptocurrency logo).
 - **Name** (Cryptocurrency name).
 - **Price** (Real-time price with sorting buttons).
 - **Volume** (Total traded volume with sorting buttons).
 - **Market Cap** (Total value with sorting buttons).
 - **Favorite** (Allows users to mark preferred cryptocurrencies).
 - **Table body (`<tbody>`):**
 - The content is **dynamically added using JavaScript**.

5. Sorting Buttons

Code →

```
<th>Price
  <i id="sort-price-asc" class="fas fa-arrow-up"></i>
  <i id="sort-price-desc" class="fas fa-arrow-down"></i>
</th>
<th>Volume
  <i id="sort-volume-asc" class="fas fa-arrow-up"></i>
  <i id="sort-volume-desc" class="fas fa-arrow-down"></i>
</th>
<th>Market Cap
  <i id="sort-market-asc" class="fas fa-arrow-up sort-icon"></i>
  <i id="sort-market-desc" class="fas fa-arrow-down sort-icon"></i>
</th>
```

Explanation:

- Users can **sort cryptocurrencies** by:
 - **Price** (Ascending/Descending).
 - **Volume** (Ascending/Descending).
 - **Market Cap** (Ascending/Descending).
- Font Awesome **arrow icons** (**fas fa-arrow-up/down**) visually indicate sorting direction.

6. JavaScript Integration (**script.js**)

Code →

```
<script src="script.js"></script>
</body>
</html>
```

Explanation:

- Connects the HTML page to an external **JavaScript file (script.js)**.
- This script handles:
 - **Fetch live data** from a cryptocurrency API.
 - **Updating the table dynamically** with real-time data.
 - **Enabling search functionality**.
 - **Handling sorting behavior**.
 - **Managing user interactions** (like marking coins as favorites).

CSS (Styling Layout)

The CSS file **Styles.css** is responsible for styling the **Cryptocurrency Dashboard** defined in the **index.html** file. It ensures a **responsive, visually appealing layout** and provides **user-friendly interactions** through navigation, search functionality, and table design.

Initiating the styles.css file as follows,

Components of CSS File:

This CSS file is responsible for styling the cryptocurrency dashboard web page. The components are detailed below as follows,

1. **Global Styles:** Uses **Lato** font, prevents horizontal scrolling, and ensures full viewport height.
2. **Navigation Bar:** Dark gray background with flexbox for alignment, highlighted links, and a hover effect.
3. **Title Styling:** Uses **bold, orange color** with a smooth transition on hover.
4. **Search Box & Dialog:** Styled with **shadows, rounded corners, and a drop-down search result area**.

5. **Main Content Area:** Uses **flexbox for layout**, ensuring a structured appearance.
 6. **Table Design:** Styled with **borders, padding, hover effects, and collapsible spacing** for readability.
 7. **Responsive Design:** Adjusts **header, navigation, and table font size for smaller screens**.
-

1. Global Styles

```
@import
url('https://fonts.googleapis.com/css2?family=Lato:wght@400;700&display=swap');
```

- Imports the **Lato** font from Google Fonts to be used throughout the page.

Body(Global) →

```
/* ===== Global Styles ===== */
body {
  font-family: 'Lato', sans-serif; /* Sets default font */
  margin: 0;
  overflow-x: hidden; /* Prevents unwanted horizontal scrolling */
  display: flex;
  flex-direction: column;
  min-height: 100vh; /* Ensures content stretches to full viewport height */
}
```

- **font-family: 'Lato', sans-serif;** → Uses Lato as the default font.
- **margin: 0;** → Removes default browser margin.
- **overflow-x: hidden;** → Prevents unwanted horizontal scrolling.
- **display: flex; flex-direction: column;** → Arranges elements in a column layout.
- **min-height: 100vh;** → Ensures the page takes up the full viewport height.

2. Navigation Bar (Header)

```
/* ===== Navigation Bar ===== */
nav {
  background-color: #333; /* Dark background for navbar */
  padding: 1em;
  display: flex;
  justify-content: space-between; /* Spacing items evenly */
  align-items: center;
  width: 100%; /* Full width */
  box-sizing: border-box; /* Includes padding in width calculation */
  position: relative; /* Ensures proper positioning of child elements */
}
```

- **background-color: #333;** → Sets a dark background for the navbar. Creates a **dark-themed navigation bar** to make it **distinct** from the rest of the content.
- **display: flex; justify-content: space-between; align-items: center;** → Distributes elements evenly and aligns them properly.
- **width: 100%;** → Makes the navbar span the full width.
- **box-sizing: border-box;** → Ensures padding doesn't affect the width.
- **position: relative;** → Makes the navbar a reference for absolute positioning of child elements.

3. Navigation Links

```
/* Navigation Links */
.nav-link {
  color: white; /* White text for contrast */
  text-decoration: none; /* Remove default underline */
  margin-right: 1em;
  background-color: #F16222; /* Orange background for buttons */
  padding: 8px 12px;
  border-radius: 5px; /* Rounded corners */
}

/* Navigation Link Hover Effect */
.nav-link:hover {
  background-color: #da561ae8; /* Darker shade on hover */
}
```

- Styles the **navigation links** to ensure they are **easily visible and interactive**.
- **White text** (`color: white;`) provides **strong contrast** against the dark navbar.
- **Removes underlines** (`text-decoration: none;`) for a **cleaner look**.
- **Orange background** (`background-color: #F16222;`) highlights the buttons.
- **Adds padding and rounded corners** to improve **visual appeal**.
- **On hover** (`:hover`), the background **darkens** slightly for a **subtle interactive effect**.

4. Left and Right Navigation Sections

```
/* Left Section of Navbar (Brand Title) */
.left-nav {
  display: flex;
  align-items: center;
}

/* Right Section of Navbar (Links & Search Box) */
.right-nav {
  display: flex;
  align-items: center;
}
```

- **Aligns items properly** in the navigation bar using **Flexbox**.
- Ensures that the **brand logo/title appears on the left** (`.left-nav`) while the **search box and links appear on the right** (`.right-nav`).

5. Website Title (Brand Name)

```
/* Website Title Styling */
.title {
  color: #F16222; /* Matching branding color */
  margin-left: auto; /* Pushes title to the right */
  text-decoration: none;
  font-weight: bold;
  font-size: 1.5rem;
  font-family: 'Arial', sans-serif;
  letter-spacing: 1px; /* Improves readability */
  transition: color 0.3s ease; /* Smooth transition effect */
}
```

```
/* Title Hover Effect */
.title:hover {
  color: #ff7643; /* Lighter shade on hover */
}
```

- Styles the **brand title/logo** to match the **branding theme**.
- **color: #F16222;** → Uses the same orange shade as the buttons for branding.
- **font-weight: bold; font-size: 1.5rem;** → Emphasizes the title.
- **letter-spacing: 1px;** → Improves readability.
- **transition: color 0.3s ease;** → Creates a smooth color transition effect.
- **Applies a hover effect** to enhance **interactivity**.

6. Search Bar Styling

```
/* ===== Search Bar Styling ===== */
.search-container {
  display: flex;
  align-items: center;
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 5px 10px;
  background-color: #f9f9f9;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); /* Subtle shadow effect */
  position: relative; /* Makes this the reference for dropdown positioning */
}
```

- Style the **search box** with a light background and subtle shadow.
- Ensures a **smooth, rounded appearance** (**border-radius: 8px;**).
- The **position: relative;** allows proper positioning of the **search dropdown**.

7. Search Dialog (Dropdown for Search Results)

```
/* Search Results Dialog Box */
#search-dialog {
  position: absolute;
  top: calc(100% + 10px); /* Positioning below the search bar */
}
```

```
left: 0;
width: 100%;
background-color: white;
box-shadow: 0 0 15px rgba(0, 0, 0, 0.5); /* Adds depth */
z-index: 1000; /* Ensures it's above other elements */
border-radius: 10px;
display: none; /* Initially hidden, shown when search is active */
}
```

- The **search results dialog box**:
 - **position: absolute; top: calc(100% + 10px); left: 0;** → Places it right below the search bar.
 - **box-shadow: 0 0 15px rgba(0, 0, 0, 0.5);** → Adds a shadow for depth.
 - **display: none;** → Hidden initially, displayed when search is triggered.
- Styles the **dropdown box for search results**.
- Ensures the **dropdown appears below the search bar**.
- Uses **shadows and a white background** for clarity.

*Search Results Content (**.dialog-content**)

```
/* Styling for Search Results Container */
.dialog-content {
  position: relative;
}
```

- **position: relative;** → Ensures that any **positioned elements** (e.g., close button) are **aligned properly** within this container.

*Header for Search Results (**.dialog-content h2**)

```
/* Header inside Search Results */
.dialog-content h2 {
  margin-left: 4px;
}
```

- **Ensures proper spacing** by adding a small left margin.
- This **header is optional** and can be removed or modified as needed.

*Close Button for Search Results (**.close-btn**)

```
/* Close Button Styling */
```

```
.close-btn {  
  position: absolute;  
  right: 20px;  
  font-size: 24px;  
  cursor: pointer;  
}
```

- **position: absolute;** → Positions the close button **inside the search results box**.
- **right: 20px;** → Moves the button to the **right side** of the container.
- **font-size: 24px;** → Makes the close button **large and easy to click**.
- **cursor: pointer;** → Changes the cursor to a **pointer on hover**, indicating interactivity.

*Search Results List (**#search-results**)

```
/* Styling for Search Results List */
```

```
#search-results {  
  list-style: none;  
  padding: 0;  
  margin: 10px 0 0;  
}
```

- **list-style: none;** → Removes default bullet points **from the list items**.
- **padding: 0;** → Ensures no extra padding **inside the list**.
- **margin: 10px 0 0;** → Adds **spacing above the first result**.

*Individual Search Result Item (**#search-results li**)

```
/* Individual Search Result Item */
```

```
#search-results li {  
  display: flex;  
  align-items: center;  
  margin: 10px 0;  
  padding: 4px 12px;  
  cursor: pointer;  
}
```

- **display: flex;** → Aligns the search result **image and text in a row**.

- **align-items: center;** → Ensures **vertical alignment** of content inside each search result.
- **margin: 10px 0;** → Provides **spacing between search results**.
- **padding: 4px 12px;** → Creates a **comfortable click area** for each result.
- **cursor: pointer;** → Indicates that **each result is clickable**.

*Hover Effect for Search Results (**#search-results li:hover**)

```
/* Search Result Item Hover Effect */
#search-results li:hover {
  background-color: #dddadb; /* Light grey background */
}
```

- **Changes background color (#dddadb)** when hovered to **highlight the selected result**.
- **Improves interactivity** by providing **visual feedback**.

*Image Styling for Search Results (**#search-results img**)

```
/* Image Styling in Search Results */
#search-results img {
  width: 32px;
  height: 32px;
  margin-right: 10px;
  border-radius: 50%;
}
```

- **width: 32px; height: 32px;** → Ensures **consistent size** for icons.
- **margin-right: 10px;** → Adds **spacing between the image and text**.
- **border-radius: 50%;** → Makes the image **circular**, enhancing the UI.

*Search Input Field (**#search-box**)

```
/* Search Box Input Field */
#search-box {
  border: none;
  padding: 4px 5px;
}
```

- **Removes border (border: none;)** for a **modern, clean look**.
- **Adds padding (padding: 4px 5px;)** to ensure text inside is **properly spaced**.

8. Main Content Area

```
main {  
  flex: 1; /* Expands to fill available space */  
  display: flex;  
  flex-direction: column; /* Stacks elements vertically */  
  align-items: center;  
  position: relative; /* Allows child element positioning */  
  gap: 10px;  
  margin-bottom: 5px; /* Prevents overlap with footer */  
}
```

- Uses **flexbox** to center elements and allow for dynamic content expansion.

9. Table Styling for Cryptocurrency Data

```
/* ===== Table Styling (Cryptocurrency Data Display) ===== */  
table {  
  width: 100%; /* Full-width table */  
  border-collapse: collapse; /* Removes spacing between borders */  
  margin-top: 20px;  
}  
  
/* Table Header & Data Cell Styling */  
th, td {  
  border: 1px solid #ddd; /* Light border for better separation */  
  padding: 8px;  
  text-align: center;  
}  
  
/* Table Header Background */  
th {  
  background-color: #f4f4f4;  
}  
  
/* Table Row Hover Effect */  
tr:hover {  
  background-color: #f1f1f1; /* Light grey highlight */  
  cursor: pointer;  
}
```

- Styles the table for displaying cryptocurrency data fetched from API.
- Uses light borders and background colors for a clean and structured look.
- Hover effect (**tr:hover**) highlights the row when hovered.

10. Responsive Design for Mobile Screens

```
/* ===== Responsive Design (For Smaller Screens) ===== */
@media (max-width: 768px) {
  /* Adjust Header Layout */
  header {
    flex-direction: column;
    text-align: center;
  }

  /* Adjust Navbar Layout */
  nav {
    margin-top: 10px;
  }

  /* Reduce Table Font Size for Mobile */
  table {
    font-size: 14px;
  }
}
```

- Ensures that the page **adapts to smaller screens** (mobile, tablets).
- **Rearranged the navbar layout** and **reduced table font size** for better readability.

Output:

CRYPTO TRACKER							Home	Liked Coins	Search Coins...
#	Icon	Name	Price ↑	Volume ↑	Market Cap ↑		Favorite		

JavaScript

This JavaScript file is responsible for fetching cryptocurrency market data from the CoinGecko API and rendering it in a table format with a favorites feature, allowing **real-time cryptocurrency data fetching, rendering in a structured table, and allowing users to favorite specific coins.**

Components of JavaScript File:

1. **API Configuration** – Defines request method and headers for fetching cryptocurrency data.
 2. **State Variables** – Stores cryptocurrency data and pagination state.
 3. **Fetching Data** – Retrieves top 25 cryptocurrencies per page from the API, with error handling.
 4. **Retrieve Favorites** – Gets saved favorite coins from `localStorage`, returning an empty array if none exist.
 5. **Render Single Row** – Creates a table row (`<tr>`) dynamically, displaying coin details and a favorite icon.
 6. **Render Coins Table** – Clears previous table data, loops through fetched coins, and appends them to the table.
 7. **Initialize Page** – Fetches cryptocurrency data and renders it on the page, handling empty data scenarios.
 8. **Execute on Load** – Ensures the script runs only after the DOM is fully loaded.
-

Components of JavaScript File:

1. API Request Configuration

```
const options = {
  method: 'GET',
  headers: {
    accept: 'application/json',
    'x-cg-demo-api-key': 'CG-mDVVqLm5xBDjvcVq523LnAmB'
  },
};
```

- Sets up the request method (`GET`) and specifies that the API should return JSON data.
- Uses a **CoinGecko API key** (`x-cg-demo-api-key`) for authentication.

2. State Variables

```
let coins = []; // Array to store coin data
let currentPage = 1; // Current page number for pagination
const itemsPerPage = 25; // Number of items per page
```

- **coins**: Stores the fetched cryptocurrency data.
- **currentPage**: Tracks the current page for pagination.
- **itemsPerPage**: Sets the number of cryptocurrencies displayed **per page** (25).

3. Fetching Coin Data from CoinGecko API

```
const fetchCoins = async (page) => {
  try {
    const response = await fetch(
      `https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=${itemsPerPage}&page=${page}`,
      options
    );
    coins = await response.json();
  } catch (err) {
    console.error(err);
  }
  return coins;
};
```

- Fetches cryptocurrency data from **CoinGecko API**.
- **Queries for market data** (price, market cap, volume) in **USD**.
- **Fetches 25 coins per page**, sorted by **market cap**.
- **Error handling**: Logs errors to the console.

4. Retrieve Favorite Coins from Local Storage(Detailed in next session)

```
const getFavorites = () => JSON.parse(localStorage.getItem('favorites')) || [];
```

- **Fetches favorited coin IDs** stored in **localStorage**.
- If no favorites exist, it **returns an empty array** ([]).
- Used to **highlight favorited coins** when rendering the table.

5. Rendering a Single Coin Row

```
const renderCoinRow = (coin, index, start, favorites) => {
  const isFavorite = favorites.includes(coin.id);
  const row = document.createElement('tr');
  row.innerHTML = `
    <td>${start + index}</td>
    <td></td>
    <td>${coin.name}</td>
    <td>${coin.current_price.toLocaleString()}</td>
    <td>${coin.total_volume.toLocaleString()}</td>
    <td>${coin.market_cap.toLocaleString()}</td>
    <td>
      <i class="fas fa-star favorite-icon ${isFavorite ? 'favorite' : ''}"
data-id="${coin.id}"></i>
    </td>`;
  return row;
};
```

- Creates a `<tr>` (table row) element dynamically.
- Includes cryptocurrency details:
 - Image, name, current price, market volume, market cap.
- Formats numbers using `.toLocaleString()` to improve readability.
- Adds a star icon (`<i class="fas fa-star">`) to mark favorites:
 - Checks if the coin is in favorites and applies a special CSS class (`favorite`).
 - Stores the coin's ID in `data-id` for event handling.

6. Rendering Coins in the Table

```
const renderCoins = (coinsToDisplay, page, itemsPerPage) => {
  const start = (page - 1) * itemsPerPage + 1;
  const favorites = getFavorites();
  const tableBody = document.querySelector('#crypto-table tbody');

  if (!tableBody) {
    console.error("Table body element not found!");
    return;
  }
}
```

```
tbody.innerHTML = ''; // Clear existing rows before rendering new data

coinsToDisplay.forEach((coin, index) => {
  const row = renderCoinRow(coin, index, start, favorites);
  tbody.appendChild(row);
});
};
```

- Clears the table (`tbody.innerHTML = ''`) before updating.
- Calculates row numbering dynamically (`start = (page - 1) * itemsPerPage + 1`).
- Retrieves favorite coins for star icon updates.
- Loops through the fetched coins and appends them as `<tr>` rows.

7. Initialize the Page

```
const initializePage = async () => {
  coins = await fetchCoins(currentPage);

  if (coins.length === 0) {
    console.error("No coins data fetched!");
    return;
  }
  renderCoins(coins, currentPage, itemsPerPage);
};
```














- Calls `fetchCoins()` to retrieve cryptocurrency data from API.
- If no data is fetched, logs an error to avoid script failure.
- Calls `renderCoins()` to display the fetched data dynamically.

8. Loading Data on Page Load

```
document.addEventListener('DOMContentLoaded', initializePage);
```

- Ensures the script runs only after the DOM is fully loaded.
- Calls `initializePage()` to fetch and display the cryptocurrency data.

Output:

CRYPTO TRACKER						
#	Icon	Name	Price ↑↓	Volume ↑↓	Market Cap ↑↓	Favorite
1		Bitcoin	\$82,252	\$37,294,671,718	\$1,632,860,372,655	★
2		Ethereum	\$2,102.46	\$23,519,914,519	\$253,620,108,118	★
3		Tether	\$1	\$27,881,618,269	\$142,792,204,981	★
4		XRP	\$2.19	\$5,946,263,020	\$126,869,550,109	★
5		BNB	\$563.38	\$1,032,434,229	\$82,229,671,552	★
6		Solana	\$127.78	\$5,068,928,117	\$65,024,044,234	★
7		USDC	\$1	\$13,827,833,297	\$58,296,591,965	★
8		Cardano	\$0.745	\$1,686,529,828	\$26,812,394,381	★
9		Dogecoin	\$0.174	\$2,126,015,753	\$25,876,874,714	★
10		TRON	\$0.234	\$883,318,952	\$22,256,762,175	★
11		Lido Staked Ether	\$2,102.39	\$91,029,557	\$19,679,996,754	★
12		Wrapped Bitcoin	\$82,142	\$466,396,135	\$10,599,242,104	★
13		Pi Network	\$1.43	\$864,148,476	\$10,327,935,600	★

Interview Questions with Answers

HTML-Based Questions

Q1. What is the purpose of the `<table>` element in the Coin Listing Page?

Answer:

The `<table>` element is used to **display cryptocurrency data in a structured format**. It consists of `<thead>` for column headings, `<tbody>` for dynamic data rows, and `<tr>` elements for each coin.

Q2. Why do we use `<meta name="viewport" content="width=device-width, initial-scale=1.0">` in the HTML file?

Answer:

This ensures the webpage is **responsive** by setting the width to the **device screen size**, preventing zoom-out effects on mobile devices.

Q3. What are semantic HTML elements used in this project?

Answer:

- `<nav>` → Defines the navigation bar.
 - `<main>` → Wraps the main content area.
 - `<table>` → Displays cryptocurrency data.
 - `<thead>` and `<tbody>` → Structure table header and body separately.
-

CSS-Based Questions

Q4. How is Flexbox used in the Coin Listing Page?

Answer:

Flexbox is used to align and distribute elements efficiently:

```
nav {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}
```

- `display: flex;` enables Flexbox.
- `justify-content: space-between;` spaces out navigation elements.
- `align-items: center;` aligns elements **vertically**.

Q5. What is the purpose of `position: relative;` in the search results dropdown?

Answer:

It ensures that **absolutely positioned elements inside (#search-dialog)** are positioned **relative to the search container** instead of the entire page.

```
.search-container {  
  position: relative;  
}
```


Q6. Why do we use `border-collapse: collapse;` in tables?

Answer:

It removes the spacing between table borders, making the table look more compact and professional.

```
table {  
  border-collapse: collapse;  
}
```

JavaScript-Based Questions

Q7. How does the Coin Listing Page fetch cryptocurrency data?

Answer:

It sends an API request using the `fetch()` function:

```
const fetchCoins = async (page = 1) => {  
  try {  
    const response = await  
fetch(`https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&page=${page}`  
), options);  
    return await response.json();  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Uses `async/await` for asynchronous API calls.
- Uses `try...catch` to handle errors gracefully.

Q8. How are favorite coins stored in the project?

Answer:

Favorite coin IDs are stored in `localStorage`, allowing users to save preferences:

```
const getFavorites = () => JSON.parse(localStorage.getItem('favorites')) || [];
```

- `localStorage.getItem('favorites')` fetches stored favorites.
- `JSON.parse()` converts the string back into an array.

Q9. How does the page ensure content loads properly?

Answer:

The script runs after the page has fully loaded:

```
document.addEventListener('DOMContentLoaded', initializePage);
```

- This ensures the API fetch and table rendering only start after the DOM is ready.

Q10. How does JavaScript update the table dynamically?

Answer:

The `renderCoins()` function creates and updates table rows dynamically:

```
const renderCoins = (coinsToDisplay, page, itemsPerPage) => {
  const tableBody = document.querySelector('#crypto-table tbody');
  tableBody.innerHTML = ''; // Clears old rows

  coinsToDisplay.forEach((coin, index) => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${index + 1}</td>
      <td>${coin.name}</td>
      <td>${coin.current_price.toLocaleString()}</td>
    `;
    tableBody.appendChild(row);
  });
};
```

- Clears previous content before inserting new rows.
- Uses `.innerHTML` to insert dynamic data.

Frequently Asked Questions (FAQs)

1. How does the Coin Listing Page retrieve real-time data?

The page **sends an API request to CoinGecko** using `fetch()`, retrieves **cryptocurrency data**, and updates the table dynamically.

2. Why is `try/catch` used in API requests?

`try...catch` **prevents script failure** by handling network errors. If the API is unavailable, an error message is logged instead of crashing the page.

3. What happens if there is no data fetched?

The script checks if data is available before rendering:

```
if (coins.length === 0) {  
  console.error("No coins data fetched!");  
  return;  
}
```

If no data is retrieved, an **error message is logged** instead of displaying an empty table.

4. How does pagination work in this project?

The script fetches coins **page by page** using:

```
const fetchCoins = async (page = 1) => {  
  return await  
  fetch(`https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&page=${page}`  
  , options);  
};
```

Users can navigate different pages by **passing different values to the page**.

5. How does the favorite feature work?

The favorite feature stores selected coins in `localStorage` and retrieves them when the page reloads:

```
const getFavorites = () => JSON.parse(localStorage.getItem('favorites')) || [];
```

- It checks if a coin is favorited and applies a `favorite` class accordingly.

6. How is responsiveness handled?

The page adapts to different screen sizes using **CSS media queries**:

```
@media (max-width: 768px) {  
  table {  
    font-size: 14px;  
  }  
}
```

This ensures the **table and navigation** adjust properly on mobile screens.

7. Can the project display more than 25 coins?

Yes, by changing `itemsPerPage`:

```
const itemsPerPage = 50; // Displays 50 coins per page
```

This will **increase the number of coins shown** before pagination is required.

8. Why do we use `innerHTML` to update the table?

`innerHTML` allows **dynamic insertion of data** inside an element:

```
tableBody.innerHTML = ''; // Clears previous rows
```

This ensures that a **fresh set of rows** is displayed each time data updates.

9. What improvements can be made to this project?

- **Sorting and filtering options** for better usability.
- **Real-time price updates** without reloading the page.
- **Graphical representation** of price trends using chart libraries.

10. How does the page know when to fetch new data?

The script **automatically fetches data** when the page loads:

```
document.addEventListener('DOMContentLoaded', initializePage);
```

Advanced Interview Tips :

1. Deep Understanding of API Handling

- Explain **asynchronous programming** using `async/await`.
- Justify the use of `try...catch` for error handling.
- Discuss **rate limits** and handling **API failures gracefully**.

2. Efficient DOM Manipulation

- Avoid excessive `innerHTML` updates; use **document fragments** for performance.
- Optimize event listeners by using **event delegation** for dynamic elements.

3. Optimizing Table Rendering

- Implement **pagination with lazy loading** instead of fetching all data at once.
- Use **debouncing** in the search functionality to reduce API calls.

4. Improving Performance & User Experience

- Store API responses in **sessionStorage/localStorage** for faster reloads.
- Implement **CSS animations or loaders** to indicate API fetching.