**Project Report**

**Name – Swaraj Santosh Pawar**

**Employee ID – 5852**

**Project On Call and Display Player’s from API in Cards**

**Problem Statement**

The project aims to develop a web application focusing on the seamless integration of an API to retrieve player information dynamically and display it in an organized manner using cards. The application should allow users to initiate an API call through a user-friendly interface, triggering the retrieval of real-time player data. The retrieved information, such as player names, statistics, and other relevant details, will be presented in visually appealing cards on the user interface. The challenge lies in efficiently handling API calls, ensuring data accuracy, and designing an intuitive and responsive card layout to enhance the user experience. Additionally, the project must address potential issues related to API connectivity and data presentation to deliver a robust and user-friendly solution for accessing and displaying player details.

**Understanding**

To implement a Player List web application, you need to create a frontend that interacts with a dummy API. First, make a call to the dummy API to retrieve player data. Once the data is fetched, store it locally. Next, design and implement a responsive user interface using cards to display the player information. Each card should represent a player and showcase relevant details. By following these steps, you can create a dynamic and visually appealing Player List application that retrieves and displays data from a dummy API.

**Introduction**

Welcome to our Player List Display project, where we embark on a journey to seamlessly integrate a dummy API, efficiently store player data, and showcase it in an aesthetically pleasing manner using responsive cards. Our goal is to create a dynamic user interface that fetches player information from the API, ensuring a smooth connection, and then organizes and presents the data within responsive cards. This project serves as a practical demonstration of API integration, data management, and responsive design, providing users with an interactive and visually appealing experience while exploring the world of player information. Join us as we delve into the realms of web development to bring you an engaging and user-friendly player list display.

**Why We Need This**

The implementation of a Player List Display with the ability to call a dummy API, store data, and present it in responsive cards serves several crucial purposes. Firstly, it allows us to showcase effective API integration, demonstrating how applications can seamlessly communicate with external data sources. This not only enhances the user experience by providing real-time and updated information but also reflects a practical approach to handling dynamic content. Additionally, by storing and organizing the player data, we ensure data persistence and accessibility, contributing to a more efficient and scalable system. The responsive design aspect addresses the diverse range of devices users may access the application from, ensuring a consistent and visually appealing experience across various screen sizes. In essence, this project amalgamates API integration, data management, and responsive design to create a robust and user-centric Player List Display, catering to the modern standards of web development.

**ALGORITHM**

1. **HTML Structure:**

* Create an HTML file with the necessary structure, including a section/div to display the player cards.

1. **CSS Styling:**

* Write CSS styles to provide a basic layout and styling for the player cards.
* Implement responsive design using media queries to adapt the layout for different screen sizes.

1. **JavaScript - Fetch Data from Dummy API:**

* Use JavaScript to make an asynchronous HTTP request to the dummy API endpoint.
* Handle the API response using promises or async/await.
* Parse the JSON data received from the API.

1. **JavaScript - Store Data:**

* Create a JavaScript data structure (e.g., an array or object) to store the player information obtained from the API.
* Populate this data structure with the parsed player data.

1. **JavaScript - Generate Cards:**

* Iterate through the stored player data.
* For each player, dynamically create HTML elements (e.g., divs) representing a card.
* Populate these card elements with the corresponding player information.

1. **JavaScript - Append Cards to HTML:**

* Select the HTML element where you want to display the player cards.
* Append the dynamically created card elements to this HTML element.

1. **JavaScript - Responsive Design :**

* Use JavaScript to dynamically apply additional styles or adjust existing styles based on the device's screen size.
* Alternatively, rely on CSS media queries for responsive design.

1. **Testing:**

* Test the application across various devices and screen sizes to ensure the player list is displayed correctly and responsively.

1. **Documentation:**

* Provide clear documentation for developers, explaining how to use and customize the code.
* Include information on the structure of the player data, API endpoint, and any other relevant details.

By following this algorithm, you can create a dynamic and responsive Player List Display using HTML, CSS, and JavaScript. This approach allows for the seamless integration of data from a dummy API, efficient data management, and a visually appealing user interface.

**Application programming interface (API)**

An API (Application Programming Interface) is a set of features and rules that exist inside a software program (the application) enabling interaction with it through software - as opposed to a human user interface. The API can be seen as a simple contract (the interface) between the application offering it and other items, such as third-party software or hardware.

In Web development, an API is generally a set of code features (e.g. methods, properties, events, and URLs) that a developer can use in their apps for interacting with components of a user's web browser, other software/hardware on the user's computer, or third-party websites and services.

**For example:**

* The getUserMedia API can be used to grab audio and video from a user's webcam, which is then available to the developer, for example, recording video and audio, broadcasting it to another user in a conference call, or capturing image stills from the video.
* The Geolocation API can be used to retrieve location information from services the user has available on their device (e.g. GPS), which can then be used in conjunction with other services, such as the Google Maps APIs, to plot the user's location on a custom map and show them what tourist attractions are in their area.
* The Web Animations API can be used to animate parts of a web page — for example, to programmatically move or rotate images.

**HTML CODE**

**Index.html**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Display Player's</title>

    <link rel="stylesheet" href="./style.css" />

    <script defer src="./script.js"></script>

    <link rel="preconnect" href="https://fonts.googleapis.com" />

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

    <link

      href="https://fonts.googleapis.com/css2?family=Ubuntu:wght@300&display=swap"

      rel="stylesheet"

    />

  </head>

  <body>

    <div class="heading">

      <h1>click button to get players</h1>

    </div>

    <button class="btn" id="btn">GET PLAYERS</button>

    <div class="main"></div>

  </body>

</html>

**CSS CODE**

body {

  background-color: #323031;

  font-family: "Ubuntu", sans-serif;

  padding: 3em;

  color: white;

}

.main {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(230px, 1fr));

  gap: 2em;

  align-items: center;

  align-content: center;

  margin-top: 18px;

}

.items {

  padding-left: 12px;

  padding-right: 12px;

  background-color: rgb(83, 161, 175);

  width: fit-content;

  border-radius: 18px;

  width: 220px;

  height: 350px;

  padding-top: 1em;

  vertical-align: baseline;

  background-image: url("https://www.svgrepo.com/show/525901/football.svg");

  background-size: cover;

  background-blend-mode: soft-light;

}

.btn {

  background-color: rebeccapurple;

  padding: 12px;

  width: 15em;

  color: white;

  border: 0;

  border-radius: 18px;

  font-size: large;

}

.btn:hover {

  background-color: blueviolet;

  cursor: pointer;

}

.head {

  display: flex;

  flex-direction: row;

  justify-content: end;

}

.summary {

  display: flex;

  flex-direction: row;

  justify-content: space-around;

}

.c {

  text-align: end;

}

.items:hover {

  transform: rotate(10deg);

  cursor: pointer;

}

::-webkit-scrollbar {

  width: 5px;

}

::-webkit-scrollbar-track {

  background: transparent;

}

::-webkit-scrollbar-thumb {

  background: rgb(83, 161, 175);

  border-radius: 5px;

}

::-webkit-scrollbar-thumb:hover {

  background: #555;

}

.heading {

  background-color: rgb(47, 72, 107);

  padding: 12px;

  border-radius: 18px;

  margin-bottom: 15px;

  text-align: center;

  text-transform: uppercase;

}

**JAVASCRIPT CODE**

**Script.js**

const URI = "https://api.npoint.io/20c1afef1661881ddc9c";

const btn = document.getElementById("btn");

const card = document.querySelector(".main");

// Fetch Api = The fetch Api provides an interface for fetching(Sending/Receiving) resources.

// It uses Request and Response Objects.

// The fetch() method is used to fetch a resource (data)

//let promise = fetch(url,[options])

// We will take data by async await

btn.addEventListener("click", async () => {

  const data = fetch(URI, {

    method: "GET",

    headers: { "Content-Type": "application/json" },

  }).then(async (res) => {

    const data = await res.json(); // we use await becz json is also Asynchronous method. We use this to make it readable.

    // Printing Cards using Loop. we have added template for player data.

    alert(" Just a moment, fetching the latest player info for you ! ");

    data.teamsList.forEach((item, i) => {

      card.innerHTML += `

                <div class="items">

                    <div class="head">${data.playerList[i].Id}</div>

                    <h3>player name</h3>

                     <h1>${data.playerList[i].PFName}</h1>

                    <div class="summary">

                        <div>

                            <h1>${data.playerList[i].Skill}</h1>

                            <h4>skills</h4>

                        </div>

                        <div class="c">

                            <h2>${

                              data.playerList[i].IsActive == 1

                                ? "ACTIVE"

                                : "INACTIVE"

                            } </h2>

                            <h4>${data.playerList[i].TName}</h4>

                        </div>

                    </div>

                    <h5>plays • ${data.playerList[i].SkillDesc}</h5>

                </div>`;

    });

  });

});

// AJAX is Asynchronous Js & XML

// JSON is JavaScript Object Notation . json format is like a javascript object format. But it is not javaScript object.

// previously data format comes in XML  Format but now it comes in json Format.

/\*

json() method : returns a Second promise that resolves with the result of parsing

the response body text as JSON.(Input is JSON, output is JS object)

 \*/

// We get response in json so we need to convert it into javaScript object. To convert it we use json() method.

**OUTPUT IMAGES**

**Image – 1**

This is the first view of the Page upon opening. It includes the Page Heading , Button and also the alert box . Example is given below.

A screen shot of a computer

Description automatically generated

.

**A screenshot of a computer

Description automatically generated**

**Image - 2**

In this section , . After Clicking the button “GET PLAYERS”, all players & their data which we fetch from API is displayed in Cards. Example is given below.

**A screenshot of a cellphone

Description automatically generated**

**A screenshot of a game

Description automatically generated**

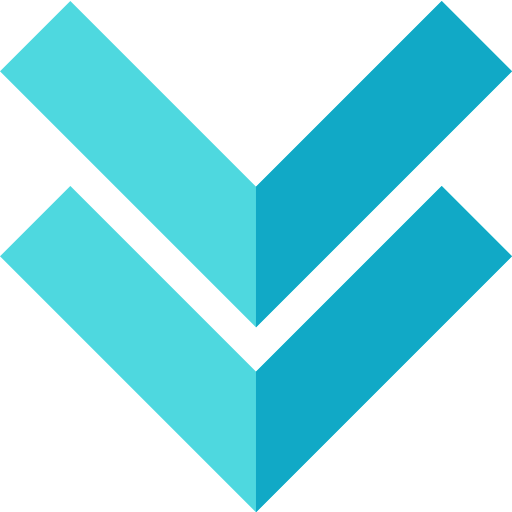
**A screenshot of a cellphone

Description automatically generated**

**A screenshot of a cellphone

Description automatically generated**

**GitHub link is given below to view Project**



**Link** - https://github.com/Swaraj-Pawar-1/api\_demo.git

**REFERENCE**

1. MDN Web Docs - <https://developer.mozilla.org/>
2. YouTube - <https://www.youtube.com/>
3. Provided API Link –“ [https://api.npoint.io/20c1afef1661881ddc9c](https://api.npoint.io/20c1afef1661881ddc9c%20) “