React Bootcamp Notes: API & Fetching Data

1. What is an API?

- **API** = Application Programming Interface
- Allows two software programs to communicate.
- Example: React app talks to a server to get data.

Example:

```
You ask: "Give me list of users"
Server responds: [ {name: "Alice"}, {name: "Bob"} ]
```

Types of APIs:

1. **REST API** – Uses HTTP methods like GET, POST, PUT, DELETE.
2. **GraphQL** – Fetch only the data you need in one query.
3. **Third-party APIs** – Example: OpenWeatherMap, Thirukkural API.

---

2. HTTP Methods

Method	Purpose
GET	Retrieve data
POST	Send new data
PUT	Update existing data
DELETE	Remove data

---

3. GET vs POST for fetching data

- **GET:** Standard method to fetch data. Sent in **URL/query string**, can be cached, idempotent.
- **POST:** Normally used to send data, but **can also fetch data** in complex scenarios.

When POST is used to fetch data:

- When you need to send **large or complex data** (filters, search queries, multiple parameters).
- When sending **sensitive info** (like tokens) in request body.
- Example: `/api/kural_adhiharam` – sending `{ adhiharam: "...", isEnglish: true }` to get filtered results.

Rule of Thumb:

- **Simple fetch → GET**
- **Complex fetch/filtering → POST**

---

4. Fetching Data in React

- Fetching = Getting data from API to show in your app.
- Usually done inside **useEffect** so it runs when the component mounts.

---

5. Fetching Data Using `fetch()`

```
import React, { useEffect, useState } from 'react';

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setUsers(data))
      .catch(err => console.log(err));
  }, []);

  return (
    <div>
      <h1>Users List</h1>
      <ul>
        {users.map(user => <li key={user.id}>{user.name}</li>)}
      </ul>
    </div>
  );
}

export default App;
```

Notes:

- `useState` stores the data.
- `useEffect` runs the fetch once on component mount.
- Always handle errors with `.catch`.

---

6. Fetching Data Using `async/await`

```
useEffect(() => {
  async function fetchData() {
    try {
      const response = await fetch('https://jsonplaceholder.typicode.com/users');
```

```
      const data = await response.json();
      setUsers(data);
    } catch (err) {
      console.error("Error fetching data", err);
    }
  }
  fetchData();
}, []);
```

- Cleaner syntax.
- `try/catch` for error handling.

---

7. POST Data to API

```
const addUser = async () => {
  const newUser = { name: "John Doe", email: "john@example.com" };

  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/users', {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(newUser)
    });

    const data = await response.json();
    console.log("Added user:", data);
  } catch (err) {
    console.error(err);
  }
};
```

- `method: "POST"` → sending data.
- `headers` → Content-Type = JSON.
- `body` → Data sent must be stringified.

---

8. Display Loading and Error States

```
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

useEffect(() => {
  async function fetchData() {
    try {
      const response = await fetch('https://jsonplaceholder.typicode.com/users');
      if (!response.ok) throw new Error("Failed to fetch");
      const data = await response.json();
      setUsers(data);
```

```
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  }
  fetchData();
}, []);

return (
  <div>
    {loading && <p>Loading...</p>}
    {error && <p>Error: {error}</p>}
    <ul>
      {users.map(user => <li key={user.id}>{user.name}</li>)}
    </ul>
  </div>
);
```

- `loading` shows spinner/text while fetching.
- `error` shows if API fails.

---

9. React + Axios (Optional)

```
import axios from 'axios';

useEffect(() => {
  axios.get('https://jsonplaceholder.typicode.com/users')
    .then(res => setUsers(res.data))
    .catch(err => console.log(err));
}, []);
```

- Axios converts JSON automatically.
- Can also POST, PUT, DELETE easily.

---

10. Key Notes for Bootcamp Students

1. **useEffect** is used to fetch data after component mounts.
2. **useState** stores API response.
3. Always handle **loading** and **error** states for better UX.
4. Use **async/await** for cleaner code.
5. **GET** → simple fetching.
6. **POST** → send data or fetch filtered/complex data.

7. For POST fetch: send **filter/search info** in request body.

8. Practice with **public APIs** like:

    - JSONPlaceholder (`https://jsonplaceholder.typicode.com/`)
    - OpenWeatherMap
    - Thirukkural API
