API INTEGRATION IN REACT - Complete Notes

Date: November 18, 2025
Time: 10:23 AM IST
Topic: Connecting React to Backend APIs
Purpose: Understanding how to fetch and send data between frontend and backend

WHAT IS API INTEGRATION?

Simple Definition:
API integration is the process of connecting your React application to a backend server to perform operations like fetching or sending data.freecodecamp+1

API stands for: Application Programming Interface

Real-world analogy:
Think of an API as a waiter in a restaurant. You (React app) tell the waiter (API) what you want, the waiter goes to the kitchen (backend server), and brings back your food (data).youtube

WHY USE APIs IN REACT?

Common Use Cases:

1. Fetch User Data
Display profile information, user lists, account detailsguvi+1

2. Authentication & Login
Verify credentials, manage sessions, handle tokensdigitalocean

3. Display Data from Database
Show products, posts, comments, articles from serverfreecodecamp+1

4. Store Form Data
Save user registrations, contact forms, survey responsesfreecodecamp

5. Real-time Updates
Get latest news, notifications, live scores

6. File Uploads
Upload images, documents, videos to server

TWO MAIN METHODS FOR API CALLS

Method 1: Fetch API (Built-in JavaScript)

What it is: Native browser feature for making HTTP requestsfreecodecamp

Advantages:

- No installation needed
- Built into JavaScript
- Works everywhere
- Lightweight

Disadvantages:

- More verbose code
- Need to manually parse JSON
- Less features than Axiosfreecodecamp

Method 2: Axios (External Library)

What it is: Popular third-party library for HTTP requestsgeeksforgeeks+1

Advantages:

- Cleaner syntax
- Automatic JSON parsing
- Better error handling
- Request/response interceptors
- More featuresdigitalocean+1

Disadvantages:

- Requires installation
- Adds to bundle size (small though)

FETCH API - COMPLETE GUIDE

Basic Syntax

```
jsx
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error))
```

Step-by-Step Fetch Example

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

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

Understanding Each Part

1. State Variables

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

- `users` stores fetched data
- `loading` tracks if request is in progress
- `error` stores any error messagesfreecodecamp

2. useEffect Hook

```jsx
useEffect(() => {
  // API call here
}, []);
```

Empty dependency array [ ] means run once when component mountsguvi+1

3. Fetch Call

```jsx
fetch('https://jsonplaceholder.typicode.com/users')
```

Sends GET request to the API endpointfreecodecamp

4. Response Handling

```jsx
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
```

- Check if response is successful
- Convert response to JSON formatfreecodecamp

5. Data Handling

```
jsx
.then(data => {
  setUsers(data);
  setLoading(false);
})
```

- Store data in state
- Set loading to falsefreecodecamp

6. Error Handling

```
jsx
.catch(error => {
  setError(error.message);
  setLoading(false);
})
```

Catch and display any errorsfreecodecamp

7. Conditional Rendering

```
jsx
if (loading) return <p>Loading...</p>;
if (error) return <p>Error: {error}</p>;
```

Show different UI based on statefreecodecamp

AXIOS - COMPLETE GUIDE

Installation

```bash
npm install axios
```

Basic Syntax

```jsx
import axios from 'axios';

axios.get('https://api.example.com/data')
  .then(response => console.log(response.data))
  .catch(error => console.error(error));
```

Step-by-Step Axios Example

```jsx
import { useState, useEffect } from 'react';
import axios from 'axios';

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        setUsers(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

Key Differences: Fetch vs Axios

Fetch:

```
jsx
fetch(url)
  .then(response => response.json())
  .then(data => setUsers(data))
```

Axios:

```
jsx
axios.get(url)
  .then(response => setUsers(response.data))
```

Notice: Axios automatically parses JSON! No need for `.json()`guvi+1

ASYNC/AWAIT SYNTAX (MODERN APPROACH)

With Fetch

```jsx
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    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);
      setLoading(false);
    } catch (error) {
      setError(error.message);
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

With Axios

```jsx
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    try {
      const response = await
axios.get('https://jsonplaceholder.typicode.com/users');
      setUsers(response.data);
      setLoading(false);
    } catch (error) {
      setError(error.message);
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

Why async/await?

- Cleaner, more readable code
- Easier to understand flow
- Better error handlingdev+1

HTTP METHODS

1. GET - Fetch Data

Purpose: Retrieve data from serverguvi+1

Fetch:

```
jsx
fetch('https://api.example.com/users')
  .then(response => response.json())
  .then(data => console.log(data))
```

Axios:

```
jsx
```

```
axios.get('https://api.example.com/users')
  .then(response => console.log(response.data))
```

2. POST - Send Data

Purpose: Create new data on serverdigitalocean

Fetch:

```
jsx
fetch('https://api.example.com/users', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'John',
    email: 'john@example.com'
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
```

Axios:

```
jsx
axios.post('https://api.example.com/users', {
  name: 'John',
  email: 'john@example.com'
})
  .then(response => console.log(response.data))
```

Notice: Axios is much simpler for POST requests!digitalocean

3. PUT - Update Data

Purpose: Update existing datadigitalocean

Axios:

```jsx
axios.put('https://api.example.com/users/1', {
  name: 'John Updated',
  email: 'john.new@example.com'
})
  .then(response => console.log(response.data))
```

4. DELETE - Remove Data

Purpose: Delete data from serverdigitalocean

Axios:

```jsx
axios.delete('https://api.example.com/users/1')
  .then(response => console.log('Deleted!'))
```

COMPLETE REAL-WORLD EXAMPLES

Example 1: Display User List

```jsx
import { useState, useEffect } from 'react';
import axios from 'axios';

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const response = await
axios.get('https://jsonplaceholder.typicode.com/users');
        setUsers(response.data);
        setLoading(false);
      } catch (error) {
        setError(error.message);
        setLoading(false);
      }
    };

    fetchUsers();
  }, []);

  if (loading) return <div>Loading users...</div>;
  if (error) return <div>Error: {error}</div>;

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

export default UserList;
```

Example 2: Create New User (POST)

```jsx
import { useState } from 'react';
import axios from 'axios';

function CreateUser() {
  const [formData, setFormData] = useState({
    name: '',
    email: ''
  });
  const [message, setMessage] = useState('');

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.post(
        'https://jsonplaceholder.typicode.com/users',
        formData
      );

      setMessage(`User created with ID: ${response.data.id}`);
      setFormData({ name: '', email: '' });
    } catch (error) {
      setMessage(`Error: ${error.message}`);
    }
  };

  return (
    <div>
      <h2>Create New User</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="name"
          placeholder="Name"
          value={formData.name}
          onChange={handleChange}
        />
        <input
          type="email"
          name="email"
          placeholder="Email"
          value={formData.email}
          onChange={handleChange}
        />
        <button type="submit">Create User</button>
      </form>
      {message && <p>{message}</p>}
    </div>
  );
}

export default CreateUser;
```

Example 3: Delete User

```
jsx
import { useState, useEffect } from 'react';
import axios from 'axios';

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

  useEffect(() => {
    fetchUsers();
  }, []);
  const fetchUsers = async () => {
    const response = await
axios.get('https://jsonplaceholder.typicode.com/users');
    setUsers(response.data);
  };

  const deleteUser = async (id) => {
    try {
      await
axios.delete(`https://jsonplaceholder.typicode.com/users/${id}`);
      setUsers(users.filter(user => user.id !== id));
      alert('User deleted successfully!');
    } catch (error) {
      alert(`Error: ${error.message}`);
    }
  };

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

export default UserListWithDelete;
```

Example 4: Search with API

```jsx
import { useState } from 'react';
import axios from 'axios';

function UserSearch() {
  const [query, setQuery] = useState('');
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(false);

  const handleSearch = async (e) => {
    e.preventDefault();
    setLoading(true);

    try {
      const response = await axios.get(

`https://jsonplaceholder.typicode.com/users?name_like=${query}`
      );
      setResults(response.data);
      setLoading(false);
    } catch (error) {
      console.error(error);
      setLoading(false);
    }
  };

  return (
    <div>
      <form onSubmit={handleSearch}>
        <input
          type="text"
          placeholder="Search users..."
          value={query}
          onChange={(e) => setQuery(e.target.value)}
        />
        <button type="submit">Search</button>
      </form>
      {loading && <p>Searching...</p>}

      <ul>
        {results.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default UserSearch;
```

ADVANCED AXIOS FEATURES

1. Base URL Configuration

Create a reusable axios instance:

jsx
```
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://jsonplaceholder.typicode.com',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
});

export default api;
```

Usage:

jsx

```
import api from './api';

api.get('/users')
  .then(response => console.log(response.data))
```

2. Request Interceptors

Add authentication tokens automatically:

```jsx
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://api.example.com'
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

export default api;
```

3. Response Interceptors

Handle errors globally:

```jsx
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response.status === 401) {
      console.log('Unauthorized! Redirect to login');
    }
    return Promise.reject(error);
  }
);
```

BEST PRACTICES

1. Always Handle Loading States

```jsx
const [loading, setLoading] = useState(false);

if (loading) return <div>Loading...</div>;
```

2. Always Handle Errors

```jsx
const [error, setError] = useState(null);

if (error) return <div>Error: {error}</div>;
```

3. Use useEffect for Data Fetching

jsx

```
useEffect(() => {
  fetchData();
}, []);
```

4. Clean Up API Calls

jsx

```
useEffect(() => {
  const controller = new AbortController();

  fetch(url, { signal: controller.signal })
    .then(response => response.json())
    .then(data => setData(data));

  return () => controller.abort();
}, []);
```

5. Store API URLs in Environment Variables

jsx

```
const API_URL = import.meta.env.VITE_API_URL;

axios.get(`${API_URL}/users`)
```

.env file:

text

```
VITE_API_URL=https://api.example.com
```

6. Create Custom Hooks

```jsx
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get(url);
        setData(response.data);
        setLoading(false);
      } catch (error) {
        setError(error.message);
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
}

function App() {
  const { data, loading, error } =
useFetch('https://api.example.com/users');

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return <div>{JSON.stringify(data)}</div>;
}
```

COMPARISON: FETCH VS AXIOS

Feature	Fetch	Axios
Built-in	Yes	No (needs install)
JSON Parsing	Manual	Automatic
Error Handling	Limited	Better
Browser Support	Modern only	All browsers
Request/Response Interceptors	No	Yes
Timeout Support	Complex	Simple
Progress Events	Limited	Yes
Code Length	Longer	Shorter

Recommendation: Use Axios for production apps, Fetch for simple projectsfreecodecamp

KEY TAKEAWAYS

1. **API Integration** connects React to backend serversguvi+1
2. **Two Main Methods:** Fetch (built-in) and Axios (library)freecodecamp
3. **useState** stores fetched data, loading state, and errorsguvi+1
4. **useEffect** makes API calls when component mountsguvi+1
5. **HTTP Methods:** GET (fetch), POST (create), PUT (update), DELETE (remove)digitalocean
6. **Axios Advantages:** Cleaner syntax, auto JSON parsing, better featuresdigitalocean+1
7. **async/await** makes code more readable than .then() chainsgeeksforgeeks+1
8. **Always Handle:** Loading states, errors, and edge casesfreecodecamp
9. **Best Practices:** Use custom hooks, environment variables, interceptorsdev+1
10. **Security:** Never expose API keys in frontend codefreecodecamp