Design a Dynamic Frontend with React

Working with APIs, JSX, and React Libraries

Engage and Think

You are browsing an e-commerce website that loads quickly and updates product availability instantly as you filter or sort. The interface responds immediately to your actions, and no full-page reloads occur during the interaction.

What design choices make this user experience feel fast and dynamic, and how can React help implement them?

Learning Objectives

By the end of this lesson, you will be able to:

- Identify the principles of RESTful architecture for better API design and implementation
- List the factors considered when choosing an API for project suitability and effectiveness
- Develop skills in making API calls using Fetch API and Axios for enhanced data interaction in React applications
- Apply the new JSX transform to enhance coding efficiency in React development
- Utilize new React libraries to manage remote data fetching effectively in web applications

API Integration and Error Handling in React

Introduction to APIs

An API allows two software systems to communicate using requests and responses.

01

Connects two applications or services

02

Sends requests and receives responses

03

Follows defined rules for data exchange

04

Works across web, mobile, cloud, and servers

05

Supports data operations like read, update, delete

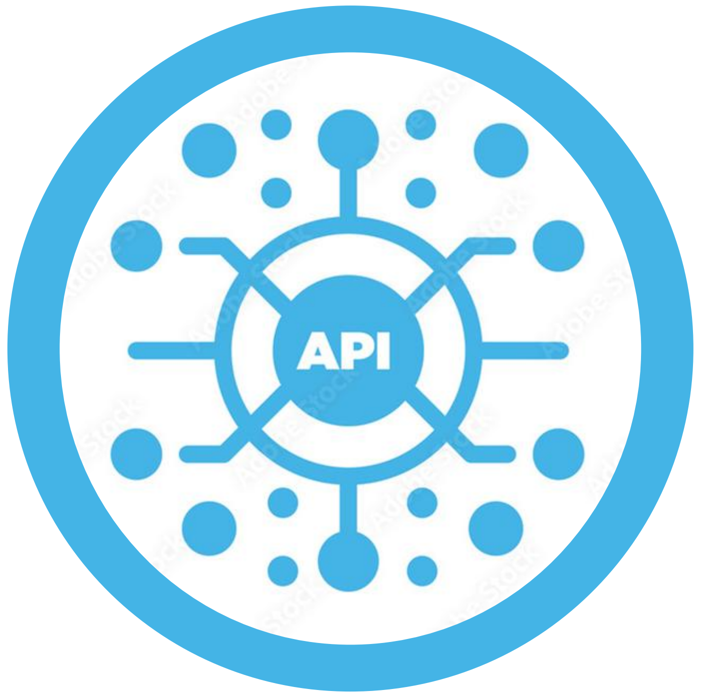
Types of APIs

APIs follow different architectural styles to define how systems communicate and exchange data.
Below are some types:

1	**REST**	A lightweight, stateless architecture widely used in web development
2	**SOAP**	A protocol-based API using XML for strict and secure communication
3	**GraphQL**	A query-based API that lets clients request specific data structures
4	**gRPC**	A high-performance, contract-based API using Protocol Buffers

RESTful API: Introduction

Representational State Transfer API (RESTful API) comprises principles for designing and developing web APIs that are scalable, flexible, and easy to maintain.



This architectural approach is widely used in constructing web services suitable for various clients, including web, mobile, and desktop applications.

How RESTful APIs Work

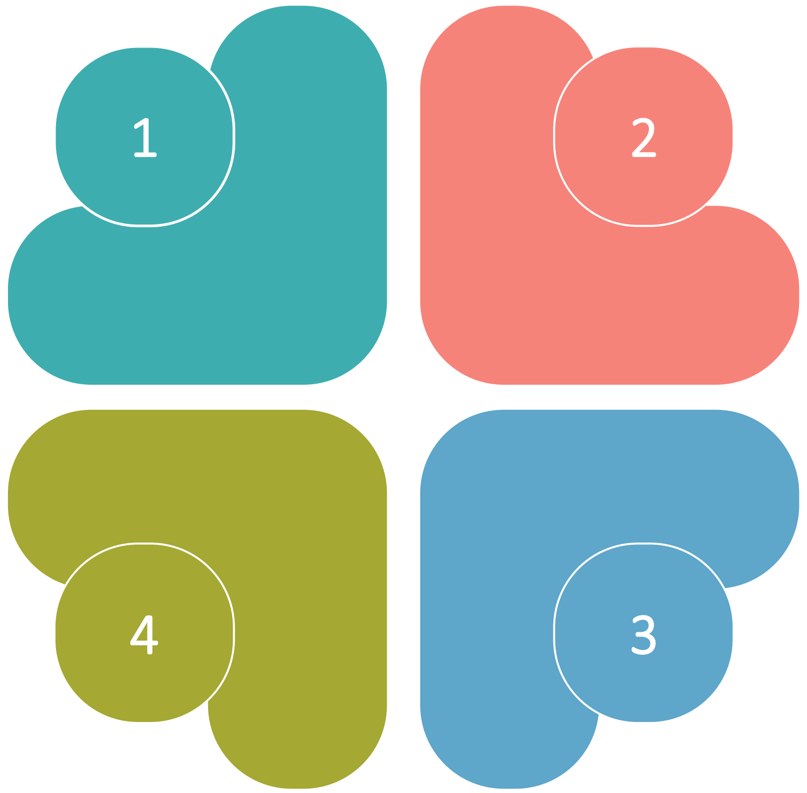
RESTful APIs provide a standardized and flexible approach that:

1. Allows communication between applications over the Internet using HTTP requests
2. Relies on REST architectural style principles for its operation
3. Performs different operations on resources using different HTTP methods
4. Returns the data using the JSON or XML formats

RESTful API: Characteristics

The characteristics of RESTful APIs are:

Strengths
Statelessness: No client data is stored on the server between requests.



Weaknesses
Client-server architecture: The client and server are separate and independent.

Opportunities
HATEOAS: Clients use links from the server to navigate the API.

Threats
Uniform interface: Standard methods are used to access resources.

Factors for Choosing an API

The following are the factors to consider while choosing an API:

- Functionality
- Documentation
- Reliability
- Scalability
- Security
- Pricing
- Assistance and support
- Data format returned by API

Factors for Choosing an API: Functionality

Functionality is a vital factor to consider when choosing an API because it:

- Meets specific needs
- Integrates seamlessly
- Supports project goals
- Streamlines development
- Enhances user experience

Factors for Choosing an API: Documentation

Documentation provides clear information about the API's features as it helps to:

- Guide API integration
- Specify API endpoints and parameters
- Outline error handling tasks
- Inform versioning
- Reduce development time
- Improve collaboration

Factors for Choosing an API: Reliability

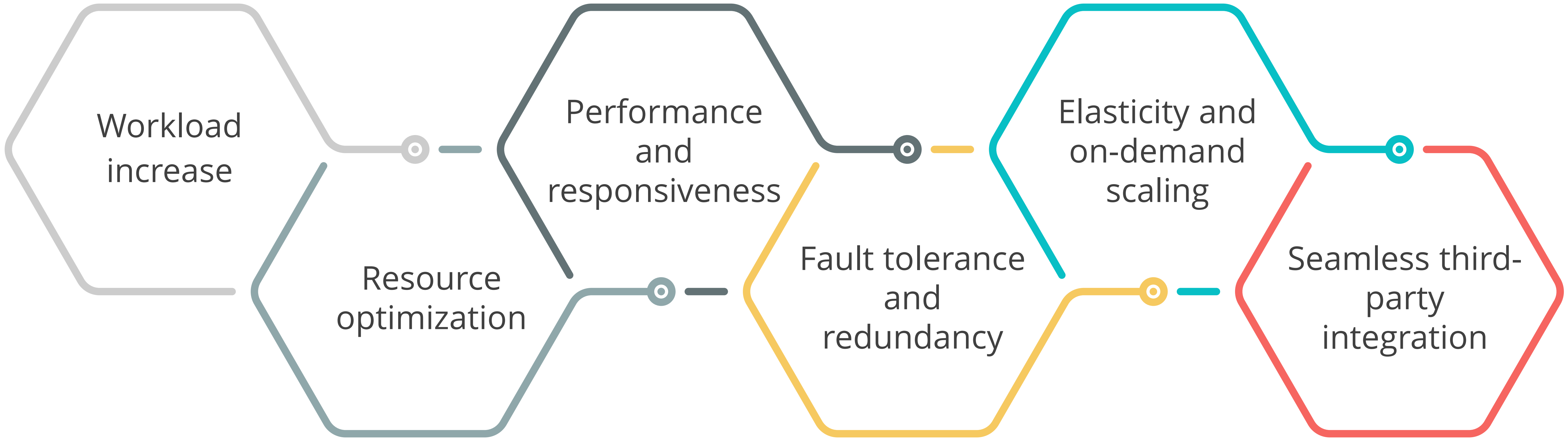
Reliability is crucial when selecting an API as it ensures:

- System stability
- Data integrity
- Robust error handling
- Predictable performance
- Service level agreements (SLAs)
- Scalability
- Efficiency and cost reduction

Factors for Choosing an API: Scalability

A scalable API can manage a growing number of users and requests.

It can also address the following:


Workload increase
Resource optimization
Performance and responsiveness
Fault tolerance and redundancy
Elasticity and on-demand scaling
Seamless third-party integration

Factors for Choosing an API: Security

Security is another vital factor when choosing an API because it protects data and user privacy.

During API implementation, security also helps in the following:

- Authentication and authorization
- Secured communication
- Vulnerability mitigation
- Compliance and regulations
- Auditing and logging

Factors for Choosing an API: Pricing

When choosing an API, it is necessary to check the pricing because it can influence the following:

- Budget management
- Cost optimization
- Scalability and growth
- Usage patterns
- Flexibility and customization
- Support and maintenance
- Value for money

Factors for Choosing an API: Support

Assistance and support facilities must be considered when selecting an API as it is an essential part of:

- API integration
- Troubleshooting and issue resolution
- Documentation updates
- Service level agreements
- Bug fixes and updates
- Vendor relationships

Factors for Choosing an API: Data Format

The data format returned by an API must be considered when choosing an API because it addresses the following:

Data compatibility

Interoperability

Data structure

Serialization efficiency

Extensibility and versioning

Standardization

Making API Calls with React

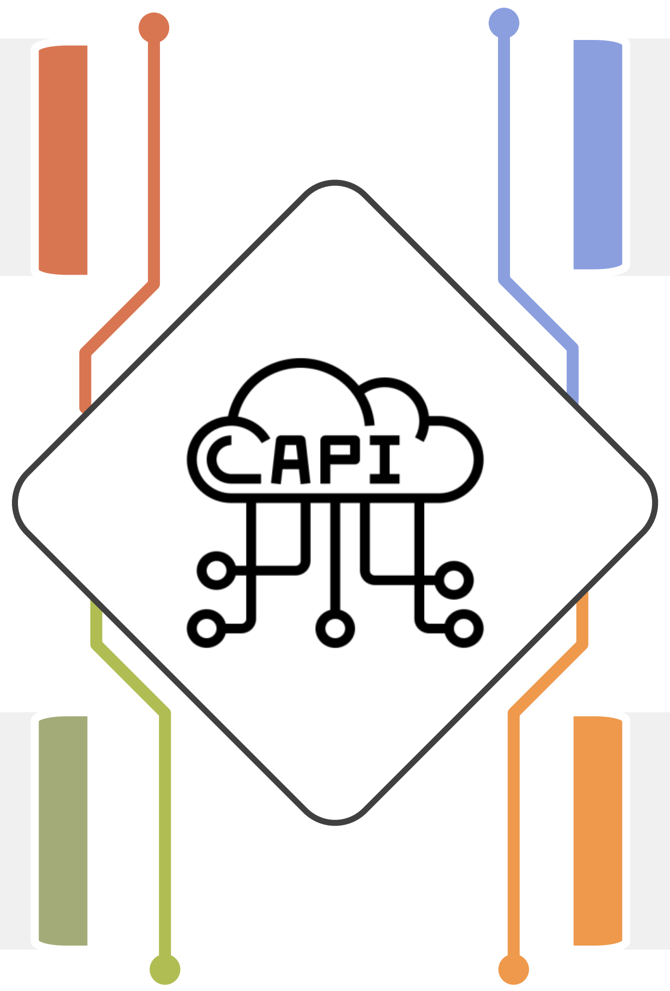
The primary purpose of making API calls in React is to build modern web applications.

There are two ways to make API calls in React:

- Using the built-in Fetch API
- Using a third-party library like Axios

Using Built-in Fetch API

The steps to make an API call using the Fetch API in React are as follows:



Construct the request object: Define the API endpoint and request options

Call the fetch() function: Use fetch() to send the API request

Handle the response: Process the returned data using .then() or async/await

Update state or perform actions: Use the response data to update UI or trigger logic

Using the Fetch API for Data Retrieval

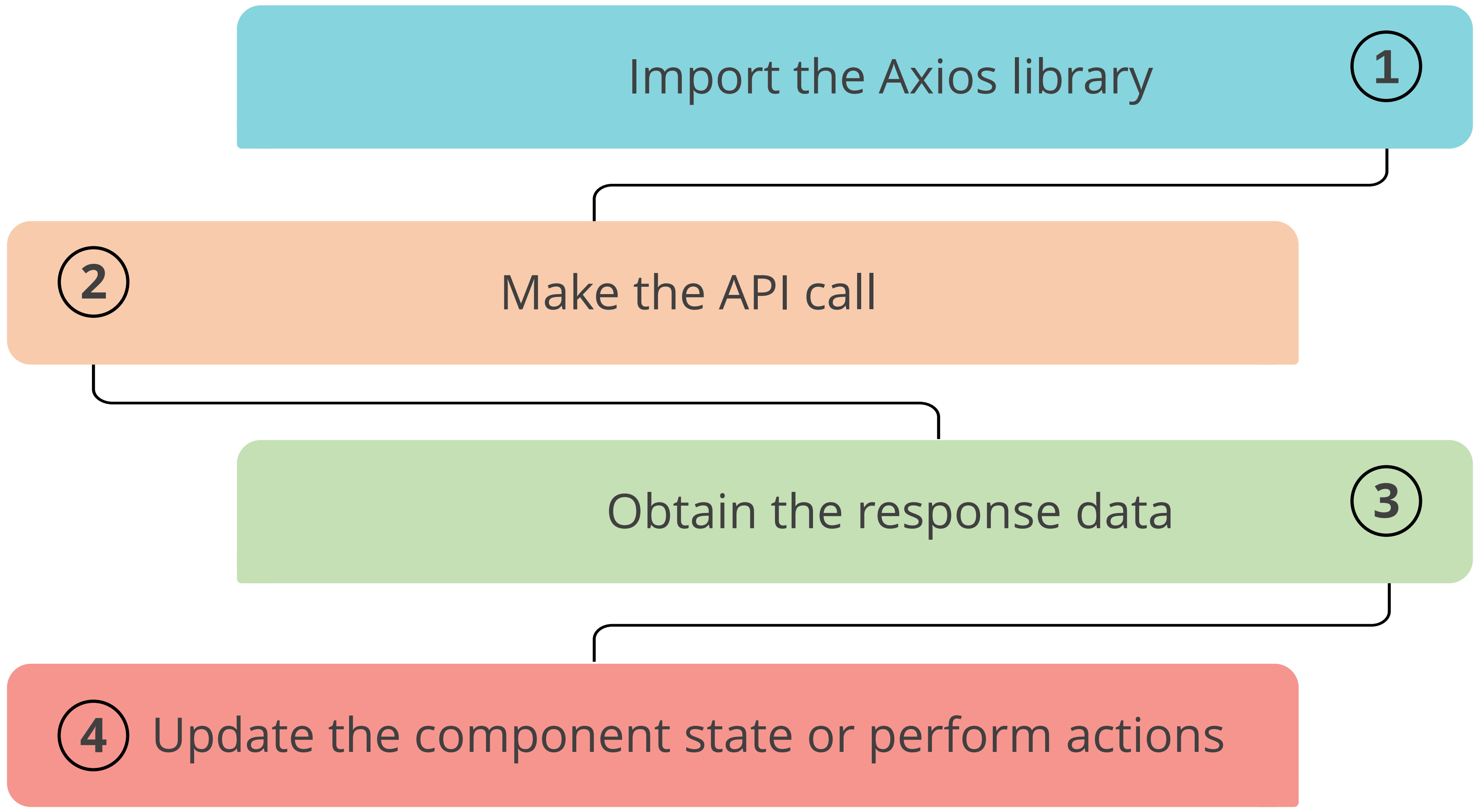
Fetching data from APIs is essential for building dynamic and interactive web applications in React.

Consider the following points while using Fetch API for data retrieval in a React application:

- Ensure that the API calls are correctly performed
- Check that the data is retrieved and displayed correctly in the React application

Making API Calls Using Axios

The steps to make an API call using Axios are as follows:



Quick Check

You are building a React application that displays real-time weather updates. To fetch the latest weather data from an external API, you need to choose the right method for making API calls in React. Which of the following options would be the most suitable for retrieving data from the API?

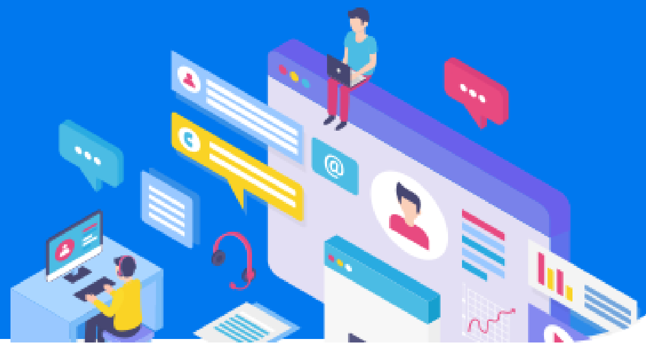
A. Fetch API and Axios

B. HTML and CSS

C. React Hooks and components

D. Local storage and session storage

Assisted Practice



JSON Response from the API **Duration: 15 Min.**

Problem statement:

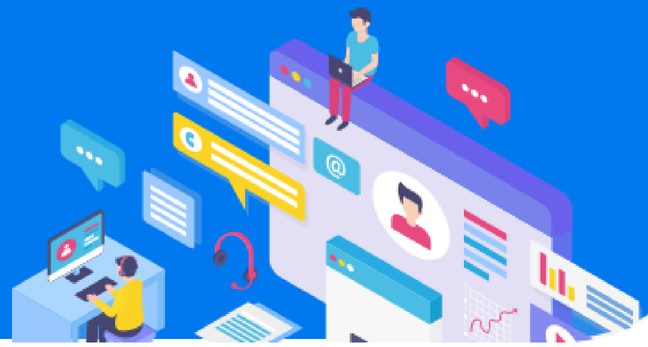
You have been assigned a task to build a simple app that displays the JSON response from the API.

Outcome:

By the end of this demo, you will be able to create a React application using Vite, build components that perform API calls, and display data dynamically using state and effect hooks.

Note: Refer to the demo document for detailed steps:
01_JSON_Response_from_the_API

Assisted Practice: Guidelines

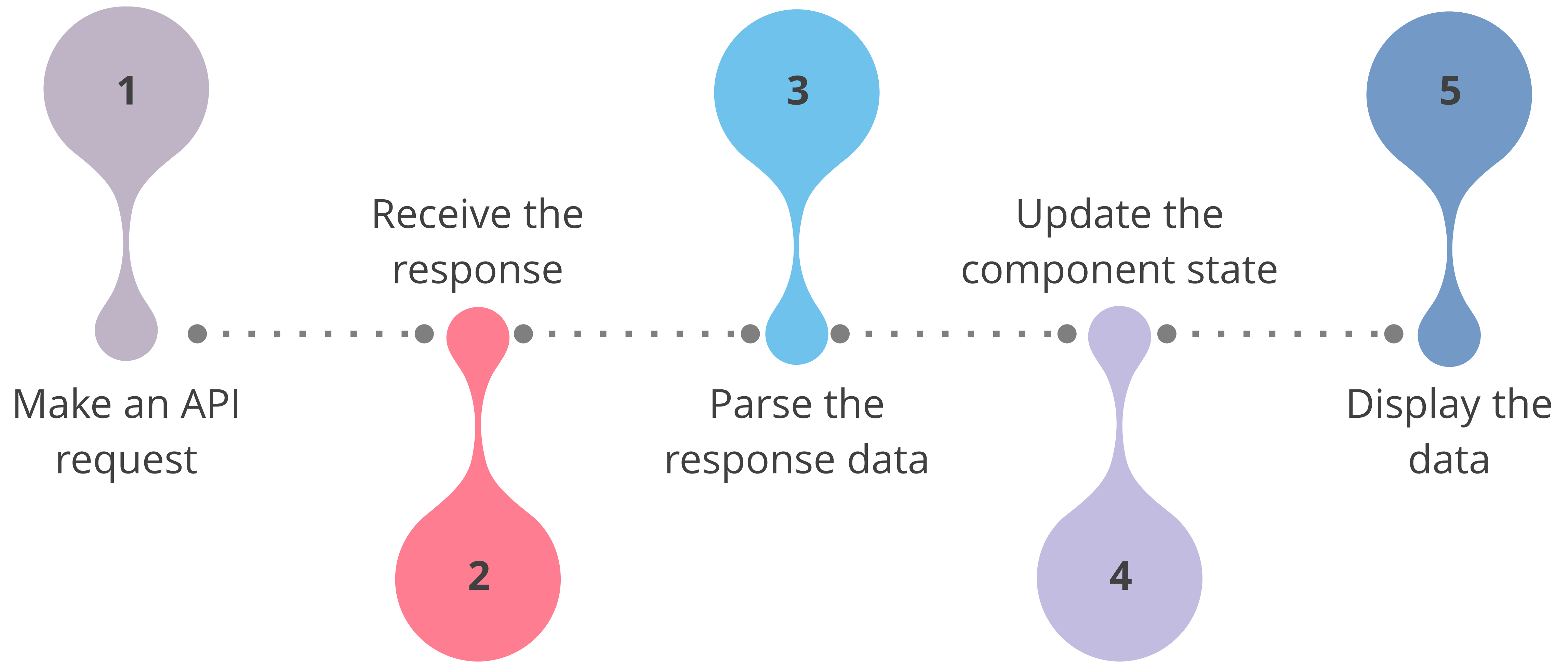


Steps to be followed:

1. Set up a new React project using Vite
2. Import and use the component
3. Run and verify the application

Steps to Handle API Responses in React

In React, API responses may include data, metadata, or error messages. Proper handling ensures accurate data display and a smooth user experience. Follow these key steps to handle API responses effectively:



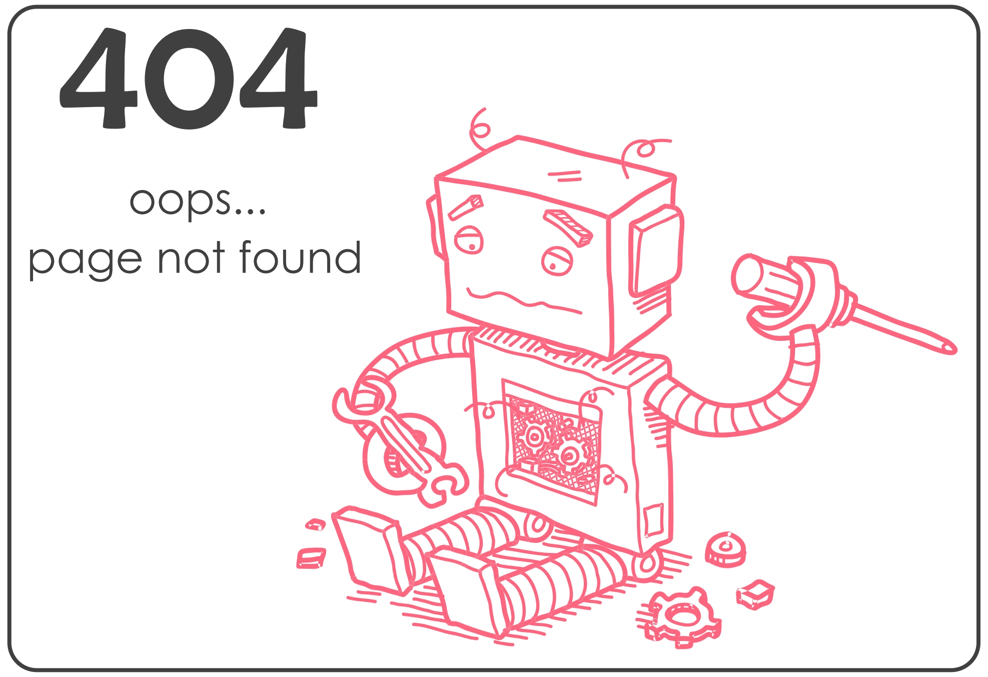
Steps to Handle API Errors in React

The steps involved in handling API errors are as follows:

1. Handle error responses
2. Extract detailed error information
3. Update error state
4. Display error message
5. Implement error-handling logic

Common HTTP Status Codes and Their Meanings

The following are commonly encountered HTTP status codes in API responses:



- **200 OK**: The request was successful.
- **201 Created**: The request was successful, and a new resource was created.
- **400 Bad Request:** The request was deformed or invalid.
- **401 Unauthorized**: The request required authentication.
- **403 Forbidden**: The server understood the request but refused to authorize it.
- **404 Not Found**: The requested resource could not be found.
- **500 Internal Server Error**: The server encountered an unexpected condition.

Using Axios to Handle API Requests and Responses

Axios is a promise-based HTTP client that helps implement API requests and responses efficiently in React applications.

Here are some reasons why Axios is popular in React programming:

- Browser compatibility
- Promise-based retrieval
- Error handling
- Compatible with interceptors
- JSON data handling
- Request cancellation
- Timeout handling

Sending Data to APIs Using Axios or Fetch

The following steps outline how to send data to an API using HTTP methods such as POST, PUT, PATCH, or DELETE:

1. Send the data to an API using HTTP methods such as **POST**, **PUT**, **PATCH**, or **DELETE**
2. Include the data in the request body and set the appropriate headers
3. Use Fetch API or Axios to send data to an API
4. Handle the response using the **then()** and **catch()** methods as needed

Axios or Fetch API helps streamline data submission, as in data retrieval, offering control over headers, request bodies, and error handling.

Quick Check

You are developing a React application that fetches user data from an API. Sometimes, the API request fails, and users see a blank screen instead of an error message. To improve the user experience, you need to implement proper error handling. What is the correct approach to handle API errors in your application?

A. Ignore the error and let the application crash

B. Log the error but do not inform the user

C. Catch the error, update the error state, and display a user-friendly message

D. Refresh the page automatically whenever an error occurs

Assisted Practice

Implementing Error Handling in API Calls **Duration: 15 Min.**

Problem statement:

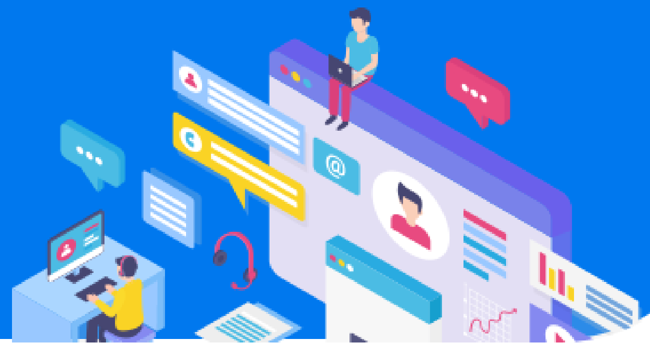
You have been assigned a task to build a simple app that displays a loading message while the API call is being made and then displays any errors.

Outcome:

By the end of this demo, you will be able to develop a React application using Vite, implement error handling in API calls, and display fallback UI to ensure a smooth user experience.

Note: Refer to the demo document for detailed steps:
02_Implementing_Error_Handling_in_API_Calls

Assisted Practice: Guidelines



Steps to be followed:

1. Set up a new React project using Vite
2. Import and use the component
3. Run and verify the application

Modern Tools for Efficient React Development

New JSX Transform in React

React 17 introduced a new JSX transform that modernizes how JSX code is compiled into JavaScript. This update provides several key benefits:

Removes the need to import React to use JSX

Improves performance and simplifies the React development process

Changes the way JSX is transformed into the JavaScript code

Enabling the New JSX Transform

The following steps enable the new JSX transform into the React components:

1 Upgrade the React project to version 17 or higher

2 Add this code to the project's babel.config.js file

```
module.exports = {
 presets: ['@babel/preset-react'],
 plugins: ['@babel/plugin-
transform-react-jsx'],
};
```

This code exports an object with the configuration options for Babel, a JavaScript compiler.

Advantages of Using the New JSX Transform

The following are the benefits of using the new JSX transform:

1 Helps improve the performance and code readability

2 Generates more optimized code

3 Offers smaller bundle sizes

4 Removes the requirement of boilerplate code

Classic vs. New JSX Transform

The differences between the classic and new JSX transform are as follows:

Classic JSX transform	New JSX transform
Requires a Babel	Does not require a Babel
Transforms JSX into React.createElement calls	Supports fragments without importing React.Fragment
Has limited optimization opportunities	Has better optimization opportunities
Produces verbose code output	Produces more concise code

Classic vs. New JSX Transform: An Example

The following code illustrates the difference between the classic JSX and the new JSX transform:

Classic JSX

```
import React from 'react';


const ClassicJSXExample = () => {
  return <input type="checkbox"
disabled={true} />;
};


export default ClassicJSXExample;
```

New JSX Transform

```
import React from 'react';


const NewJSXTransformExample = () => {
  return <input type="checkbox" disabled
/>;
};


export default NewJSXTransformExample;
```

The difference lies in Boolean handling: classic JSX uses disabled={true}, while the new transform supports shorthand like disabled, making code cleaner.

Limitations of the New JSX Transform

The following are the issues and limitations of the new JSX transform:

1. The new transform is not compatible with the older versions of the React app.
2. It may present some edge cases with certain combinations of conditional rendering.
3. Developers must test code before switching, as some third-party libraries may not be compatible.

Limitations of the New JSX Transform: An Example

```
import React from 'react';
// Custom Component
const MyComponent = ({ isVisible }) => {
  return isVisible ? <div>Visible</div> :
null;
};
const NewJSXTransformExample = () => {
  const isVisible = false;
  return <MyComponent
isVisible={isVisible} />;
};
export default NewJSXTransformExample;
```

The code conditionally renders the MyComponent component based on the Boolean prop isVisible, showcasing the new JSX transform syntax.

Limitations of the New JSX Transform: An Example

```
const NewJSXTransformExample = () => {
  const isVisible = false;
  return <MyComponent
isVisible={isVisible} />;
};
```

The code passes a Boolean prop named isVisible to MyComponent, enabling internal conditional rendering using the new JSX transform.

Quick Check

You are working on a React project and notice that the current JSX code requires importing React in every file, making the codebase bulky. You want to improve performance and reduce unnecessary boilerplate code. Which feature can help achieve this?

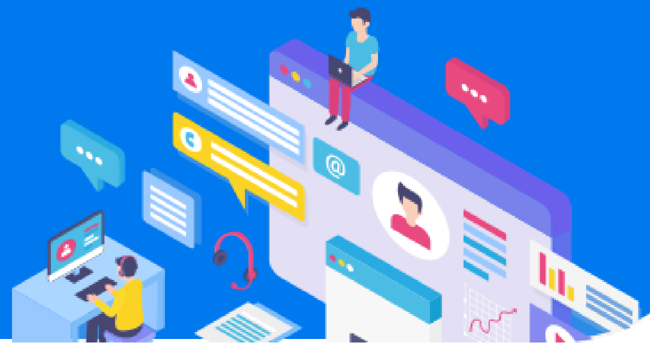
A. Using inline styles instead of CSS files

B. Manually optimizing the JavaScript bundle

C. Enabling the new JSX transform

D. Removing all JSX from the project

Assisted Practice



Creating a React App Using JSX Transform **Duration: 15 Min.**

Problem statement:

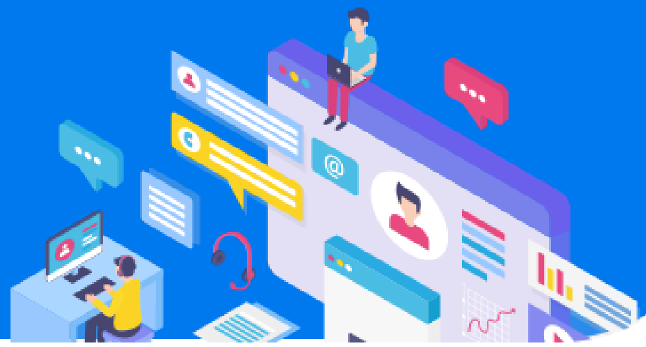
You have been assigned to build an app that displays a heading and a paragraph using the new JSX transform.

Outcome:

By the end of this demo, you will be able to develop a React application using Vite, apply the new JSX transform, and simplify your workflow by removing the need for explicit React imports.

Note: Refer to the demo document for detailed steps:
03_Creating_a_React_App_Using_JSX_Transform

Assisted Practice: Guidelines



Steps to be followed:

1. Set up a new React project using Vite
2. Import and use the component
3. Run and verify the application

Popular New Libraries in the React Ecosystem

The React ecosystem continues to evolve with libraries that enhance development efficiency. Two widely adopted libraries that support modern React applications are:

React Query helps simplify remote data fetching and caching.

React Hook Form helps build performant and flexible forms.

Key Benefits of Using React Query

React Query simplifies and enhances the process of fetching, caching, and updating remote data in React applications.

It offers the following benefits:

- Eliminates the complexity of managing data-fetching logic manually
- Provides a declarative and intuitive API for component-level data management
- Supports advanced features like caching, optimistic updates, background refetching, and polling
- Integrates seamlessly with any data-fetching library, including Axios and Fetch API

Benefits of React Hook Form

React Hook Form simplifies form handling in React applications and provides the following benefits:

- Adopts a minimalist design with a simple and intuitive API
- Minimizes re-renders to boost form performance
- Supports advanced features such as conditional inputs and custom error messages

Quick Check

You are developing a React application that requires handling user input through dynamic forms while also fetching data from a remote API efficiently. You need a solution that minimizes re-renders, improves performance, and caches API responses to reduce unnecessary network requests. Which combination of React libraries would best help you achieve these functionalities?

A. React Query and React Hook Form

B. Redux and React Router

C. Bootstrap and Material UI

D. jQuery and Lodash

Assisted Practice

Integrating React Libraries: Axios and Formik **Duration: 15 Min.**

Problem statement:

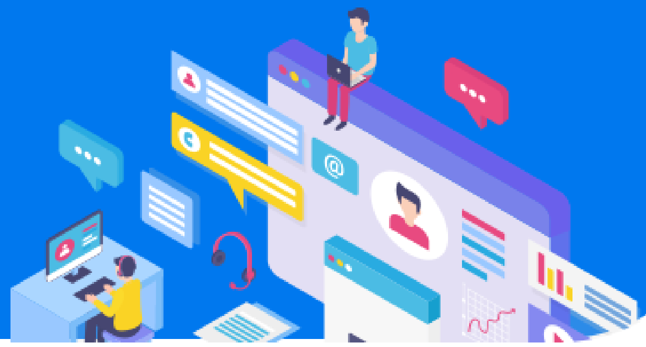
You have been assigned to build a simple form that allows submitting a new post to a remote server with form validation and state management using Axios and Formik.

Outcome:

By the end of this demo, you will be able to create a React application that uses Formik for form validation and Axios for submitting data to an API, enabling efficient form state management and remote data handling.

Note: Refer to the demo document for detailed steps:
04_Integrating_React_Libraries- Axios_and_Formik

Assisted Practice: Guidelines



Steps to be followed:

1. Set up a new React project using Vite
2. Import and use the component
3. Run and verify the application

Building a Blog Dashboard with React and Vite

Duration: 25 Min.

Project agenda: The goal is to develop a modern blog dashboard application using Vite and React. The app will allow users to fetch and display blog posts from an external RESTful API and submit new posts using a form. This project will demonstrate the use of Fetch API, Axios, React Query, React Hook Form, error handling, and the new JSX transform.

Description: You are tasked with building a React-based blog dashboard to enhance performance, interactivity, and modern development practices. This involves using Vite for fast bundling, implementing RESTful API integration using Fetch and Axios, managing asynchronous data with React Query, and handling user input with React Hook Form. The project ensures a seamless user experience through proper error handling, real-time updates, and an intuitive interface powered by the new JSX transform.

Building a Blog Dashboard with React and Vite

Steps to be performed:

1. Set up a new React project using Vite
2. Create and connect blog post components
3. Modify App.jsx file
4. Update main.jsx file
5. Run and verify the application

Expected deliverables: A fully functional React-based blog dashboard application with core features, including dynamic data fetching and submission using RESTful APIs. The application will support viewing posts using React Query, submitting new posts via Axios and React Hook Form, and providing UI feedback through error and loading states. It will be structured with reusable components, efficient data handling using hooks, and a clean, intuitive user interface powered by the new JSX transform.

Key Takeaways

- An API allows two software systems to communicate using requests and responses. SOAP offers a more secure but heavyweight structure often used in enterprise systems.
- GraphQL gives more control over the retrieved data, making it popular in modern web apps. gRPC is great for high-performance communication between microservices.
- Axios provides a simple and easy-to-use API for working with APIs in a React application.
- The new JSX transform is a feature introduced in React version 17 that changes the way JSX is transformed into JavaScript code.
- React Query is designed to simplify fetching and caching remote data in React applications.

Thank You