ReactJS

TECHNOLOGY

API Calling, JSX, and React Libraries

simplilearn

Learning Objectives

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

- Identify the principles of RESTful architecture
- List the factors that are considered while choosing an API
- Develop skills in making API calls using Fetch API and Axios
- Work with the new JSX transform
- Use the new React libraries for managing remote data fetching

A Day in the Life of a ReactJS Developer

Sarah was hired to solve an issue in a website. She detected that the React code scripted for the website could not provide seamless connectivity. After careful analysis, she identified the following solutions:

- Axios should be used for asynchronous HTTP requests to the REST endpoints.
- Stringent technologies, such as Axios and JSX, should be used to prevent data forgery.

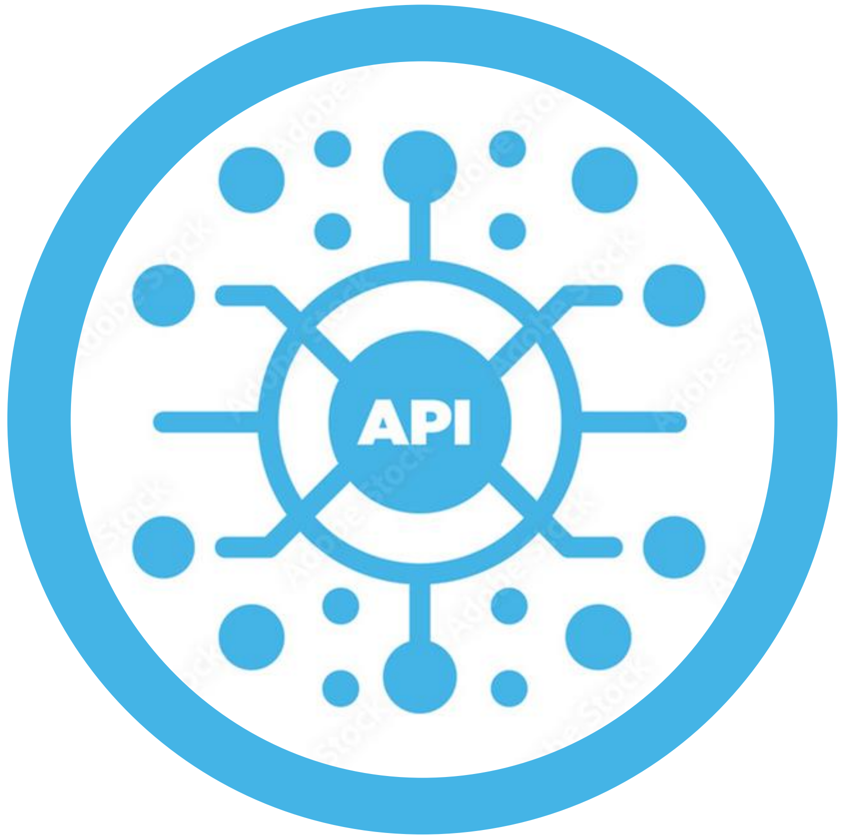
The following slides deal with the API calls using Axios and the role of JSX in making the scripting of the code easier and faster.

TECHNOLOGY

API Calling with React and Axios, Handling API Response, and Errors

Understanding RESTful API

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



It is a popular architectural approach used for constructing web services that can be consumed by diverse clients, including web, mobile, and desktop applications.

RESTful API: Characteristics

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 core principles of RESTful APIs are:

- Statelessness
- Client-server architecture
- Uniform interface
- Hypermedia as the engine of application state (HATEOAS)

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

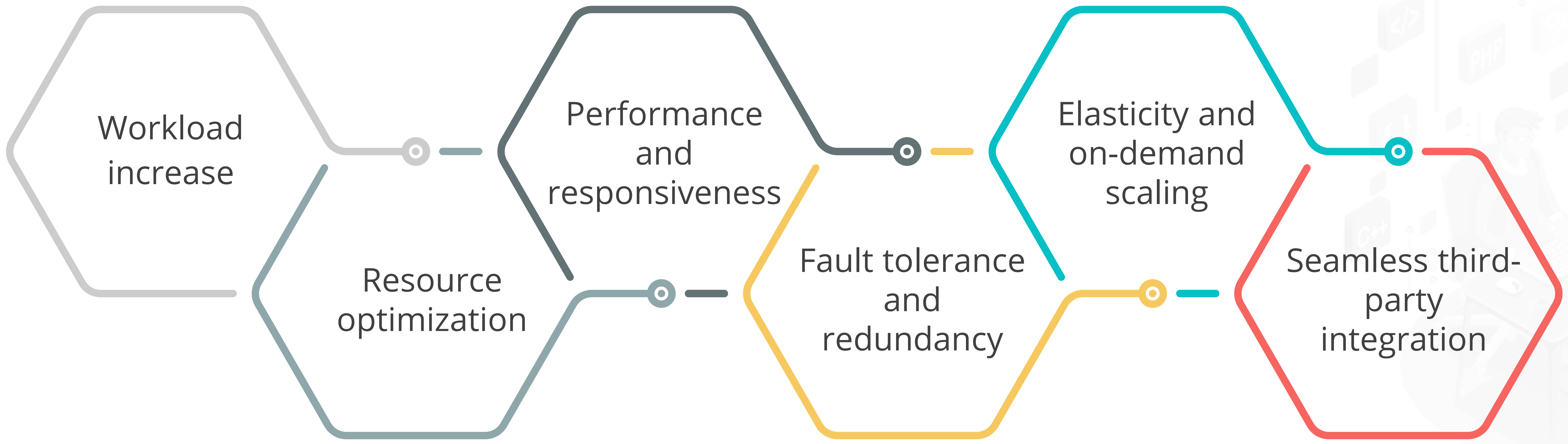
Factors for Choosing an API: Reliability

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

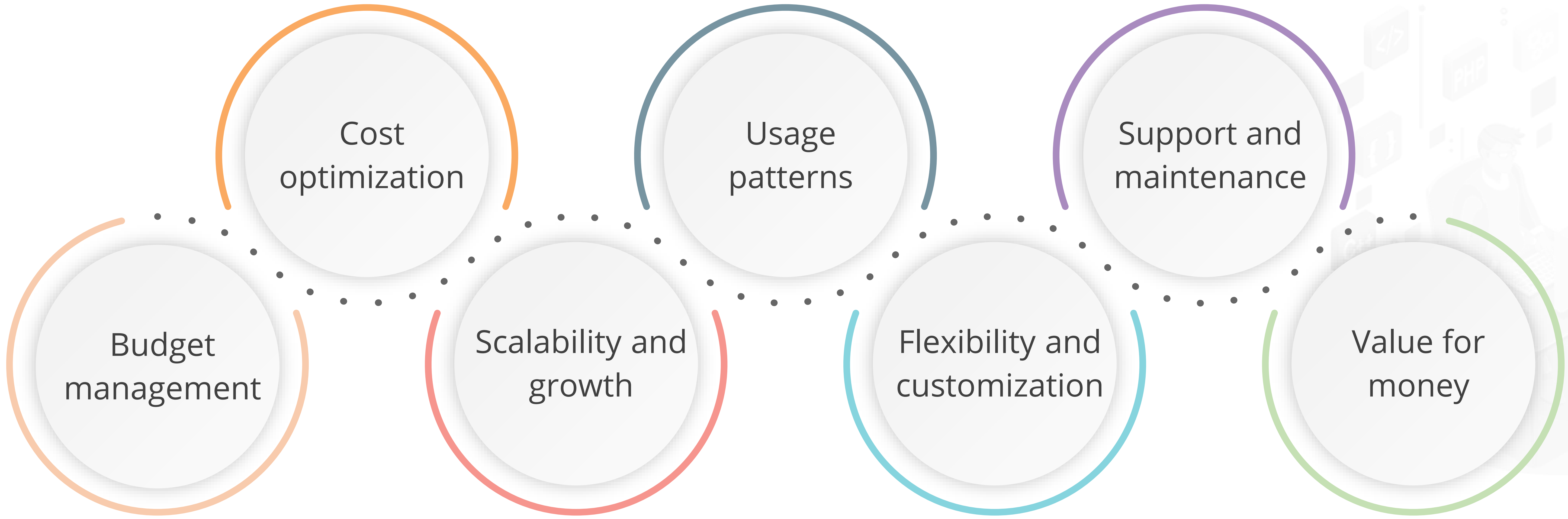
Security is another vital factor when choosing an API because it protects data and user privacy.

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:



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 (SLAs)
- 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:

1. Construct the request object
2. Call the **fetch()** function
3. Handle the response
4. Update the component state or perform actions

Making API Calls using Axios

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

1. Import the Axios library
2. Make the API call
3. Obtain the response data
4. Update the component state or perform actions

Using the Fetch API for Data Retrieval

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

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

Demo with API Calls with React

Duration: 30 Min.

Problem Statement:

You are given a project to build a simple app that displays the JSON response from the API.

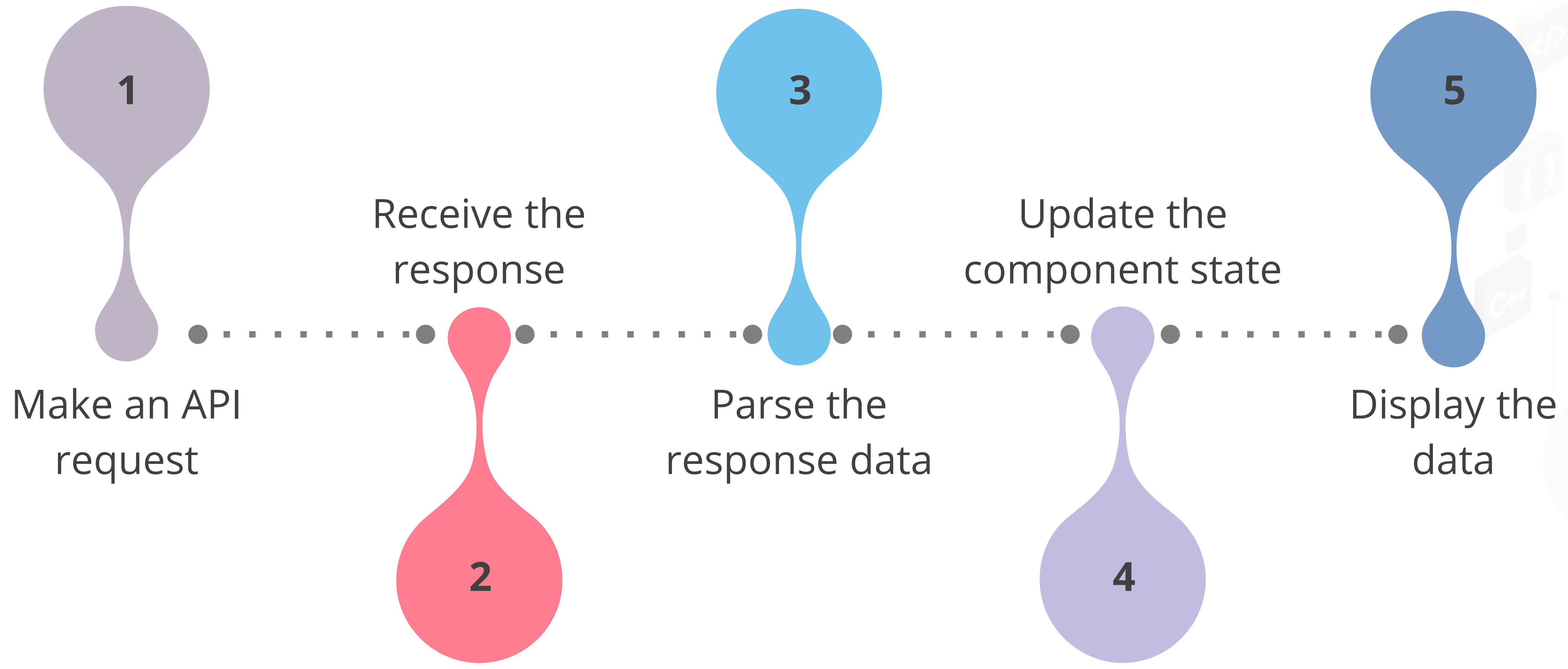
Assisted Practice: Guidelines

Steps to be followed:

1. Create a new React app using **create-react-app**
2. Change to the newly created directory by running **cd api-demo**
3. Open the project in your preferred code editor
4. Create a new file called **App.js** in the src directory
5. In **App.js**, define a state variable called data and a function called **fetchData**
6. Employ the **useEffect** hook to call **fetchData** when the component mounts
7. Run the app by running **npm start** in your terminal
8. Open **http://localhost:3000** in your browser

Handling API Responses and Errors

In React, API responses may include data, metadata, or error messages.
Here are the steps involved in handling API responses:



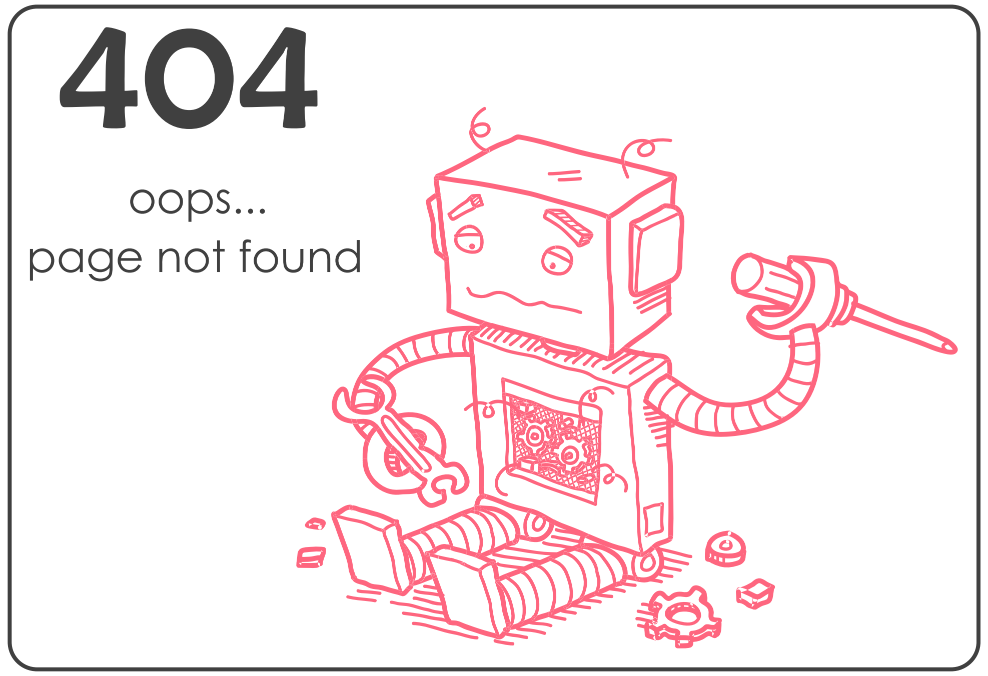
Handling API Errors

The steps involved in handling API errors are as follows:

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

Handling API Errors

The most common HTTP status code and their meanings are:



- 200 OK: The request was effective.
- 201 Created: The request was effective, and a new resource was built.
- 400 Bad Request: The request was deformed or not valid.
- 401 Unauthorized: The request requires authentication.
- 403 Forbidden: The request is not allowed.
- 404 Not Found: The requested resource could not be found.
- 500 Internal Server Error: An error occurred on the server.

Using Axios for Data Retrieval

Axios can be used to retrieve data from an API by providing the API endpoint and any required parameters or headers.

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 with API Calls

The following steps must be implemented when sending data to an API using the HTTP method:

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

In this method, the Fetch API or Axios library is used.

Demo with Error Handling

Duration: 15 Min.

Problem Statement:

You are given a project to build a simple app that displays a loading message while the API call is being made, and then displays any errors.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new React app using **create-react-app**
2. Change to the newly created directory by running **cd error-demo**
3. Open the project in your preferred code editor
4. Create a new file called **App.js** in the src directory
5. In **App.js**, define a state variable called error, and a function called **fetchData**
6. Use a **try-catch** block to catch any errors that occur during the API call
7. Run the app by running **npm start** in your terminal
8. Open **http://localhost:3000** in your browser

Support for the New JSX Transform, and Newly Launched Libraries

New JSX Transform in React

The JSX transform is a new feature introduced in React version 17, and has the following features:

Changes the way JSX is transformed into the JavaScript code

Improves performance and simplifies the React development process

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'],
};
```

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 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 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 the handling of boolean attributes.

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. The new transform 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 renders the **MyComponent** component conditionally based on the boolean prop **isVisible**, demonstrating the usage of the new JSX transform.

Limitations of the New JSX Transform: An Example

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

The code renders the **MyComponent** component conditionally based on the boolean prop **isVisible**, demonstrating the usage of the new JSX transform.

Demo with New JSX Transform in React

Duration: 15 Min.

Problem Statement:

You are given a project to build an app that displays a heading and a paragraph using new JSX transform.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new React app using **create-react-app**
2. Change into the newly created directory by running **cd jsx-demo**
3. Open the project in your preferred code editor
4. In **App.js**, create a simple function component that uses JSX to render some elements
5. Run the app by running **npm start** in your terminal
6. Open **http://localhost:3000** in your browser

New React Libraries

The React ecosystem is constantly evolving and has introduced two new React libraries that can be used to build React applications.

React Query helps simplify remote data fetching and caching.

React Hook Form helps build performant and flexible forms.

Benefits of React Query

React Query simplifies the process of fetching data from APIs.

It offers the following benefits:

- Helps remove the complexities of fetching data from APIs
- Offers a declarative and intuitive API for managing data in components
- Provides advanced features such as caching, optimistic updates, and polling
- Shares compatibility with any data fetching library like Axios and Fetch API

Benefits of React Hook Form

React Hook Form offers the following benefits:

- Follows a minimalist approach, and provides a simple API
- Reduces the number of re-renders
- Supports advanced features such as conditional form inputs and error messages

Demo with Use of React Libraries

Duration: 15 Min.

Problem Statement:

You are given a project 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.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new React app using **create-react-app**
2. Change into the newly created directory by running **cd react-libraries-demo**
3. Open the project in your preferred code editor
4. Install two popular React libraries: Axios for remote data fetching, and Formik
5. Run **npm install axios formik** in your terminal
6. In **App.js**, import Axios and Formik, and create a simple form
7. Run the app by running **npm start** in your terminal
8. Open **http://localhost:3000** in your browser

Key Takeaways

- RESTful API is an architectural style used for building web services that can be consumed by various clients.
- There are two ways to make API calls in React.
- The new JSX transform is a feature in React that allows developers to use the new syntax, such as optional chaining.
- The advantages of the new JSX transform include improved developer experience, better readability, and code conciseness.
- React Query is a library for fetching data that provides a simpler and more efficient alternative to other data-fetching libraries.
- React Hook Form is a library for building flexible forms that support minimal setup and the use of React hooks.

Lesson-End Project: Using Axios Methods

Duration: 30 Min.

Problem Statement:
Demonstrate how to use **Axios** to fetch data from an API in a React component using the **useEffect** Hook

Steps to be performed:

1. Create a new React project using **create-react-app** or any other method you prefer
2. Install Axios in your project using **npm install Axios**
3. Create a new file called **UserList.js**
4. In **UserList.js**, import React and Axios
5. Create a state variable to store the list of users
6. In the **useEffect** Hook, use Axios to make an HTTP request to an API that returns a list of users
7. Set the state variable to the response data
8. Render the list of users using **map()**
9. In your **App.js** file, import **UserList.js**
10. Render the **UserList** component

LESSON-END PROJECT

TECHNOLOGY

Thank You