React Js – Component Life Cycle

* componentWillMount :- is executed before rendering, on the server and the client side.
* componentDidMount:- is executed after the first render only, on the client side.

setTimeout or setInterval

* componentWillReceiveProps :- is invoked as soon as the props are updated before another render is called.

setNewNumber :- is used to update the state.

* shouldComponentUpdate :- this is returning true or false value. This is to check whether the component will be updated on every state and props data.
* componentWillUpdate : is called just before rendering.
* componentWillUnmount :- is called after the component is unmounted from the dom.

# Difference between fetch and axios.js for making http requests

# One of the fundamental tasks of any web application is to communicate with servers through the HTTP protocol. This can be easily achieved using Fetch or Axios. Fetch and Axios are very similar in functionality. Some developers prefer Axios over built-in APIs for its ease of use. The Fetch API is perfectly capable of reproducing the key features of Axios.

[Fetch:](https://www.geeksforgeeks.org/fetch-api/) The Fetch API provides a **fetch() method** defined on the window object. It also provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline (requests and responses). The fetch method has one mandatory argument- the URL of the resource to be fetched. This method returns a Promise that can be used to retrieve the response of the request.

# fetch('path-to-the-resource-to-be-fetched')

# .then((response) => {

# // code for handling the response

# })

# .catch((error) => {

# // code for handling the error

# });

# Axios: Axios is a Javascript library used to make HTTP requests from node.js or XMLHttpRequests from the browser and it supports the Promise API that is native to JS ES6. It can be used intercept HTTP requests and responses and enables client-side protection against XSRF. It also has the ability to cancel requests.

# axios.get('url')

# .then((response) => {

# // Code for handling the response

# })

# .catch(error) => {

# // Code for handling the error

# })

**Differences between Axios and Fetch:**

|  |  |
| --- | --- |
| Axios | Fetch |
| Axios has **url** in request object. | Fetch has **no url** in request object. |
| Axios is a **stand-alone third party package** that can be easily installed. | Fetch is built into most modern browsers; **no installation** is required as such. |
| Axios enjoys built-in XSRF protection. | Fetch does not. |
| Axios uses the **data** property. | Fetch uses the **body** property. |
| Axios’ data contains the **object**. | Fetch’s body has to be **stringified**. |
| Axios request is ok when **status is 200** and **statusText is ‘OK’**. | Fetch request is ok when **response object contains the ok property**. |
| Axios performs **automatic transforms of JSON data**. | Fetch is a **two-step process** when handling JSON data- first, to make the actual request; second, to call the .json() method on the response. |
| Axios allows **cancelling request and request timeout**. | Fetch does not. |
| Axios has the ability to **intercept HTTP requests**. | Fetch, by default, doesn’t provide a way to intercept requests. |
| Axios has **built-in support for download progress**. | Fetch does not support upload progress. |
| Axios has **wide browser support**. | Fetch only supports Chrome 42+, Firefox 39+, Edge 14+, and Safari 10.1+ (This is known as Backward Compatibilty). |

# HOW TO CONSUME A RESTFUL API IN REACT

In this session we are going to understand how to integrate a RESTful API in a React application.

React is the one of the most popular frontend framework out there and we developers are increasing learning how to build real life applications with React. While learning React, we will eventually get to a point when we need to integrate APIs in your React application.

We will be building a simple contact list application to display the contact's name, email and we will store a catch phrase of a contact. We will make use of this [endpoint](http://jsonplaceholder.typicode.com/users), it contains a JSON dump of data needed for our contact list application.

npx create-react-app react-api

### Setting up bootstrap

link bootstrap's CDN in the index.html file

<head>

...

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

...

</head>

When this is done we will render a bootstrap card in the App.js file by including this snippet in the return() method.

*// src/App.js*

import React, { Component } from 'react';

class App extends Component {

render() {

return (

<div class="card">

<div class="card-body">

<h5 class="card-title">Steve Jobs</h5>

<h6 class="card-subtitle mb-2 text-muted">steve@apple.com</h6>

<p class="card-text">Stay Hungry, Stay Foolish</p>

</div>

</div>

);

}

}

export default App;

You will need npm installed, and a basic knowledge of React.

This brief tutorial will help you understand a few concepts you need to know so as to integrate a RESTful API in a React application.

React is the one of the most popular frontend framework out there and more developers are increasing learning how to build real life applications with React. While learning React, you will eventually get to a point when you need to integrate APIs in your React application.

We will be building a simple contact list application to display the contact's name, email and we will store a catch phrase of a contact. We will make use of this [endpoint](http://jsonplaceholder.typicode.com/users), it contains a JSON dump of data needed for our contact list application.

## Prerequisites

* Basic knowledge of React (versions >=2.1.1)
* Basic understanding of JavaScript
* Basic understanding of APIs
* Must have npm installed on your local machine (versions >= 5.2)

## Setting up our environment

### Install create-react-app

We will be creating our React Application using a neat tool called create-react-app, this is a boilerplate that helps you set up a proper folder structure and assets binding for your React application. Run this command on your terminal.

npm i create-react-app

### Create a new Project

Next, we will create a new project using create-react-app. So you can switch into your preferred directory and run this command on your terminal.

npx create-react-app react-api

npx is a tool to help execute packages, you can read more about it [here](https://www.npmjs.com/package/npx)

This will generate some scaffolds you need to get your React application running as quick as possible.

Next, run the application using the following command.

npm start

This will automatically open [http://localhost:3000](http://localhost:3000/) on your browser and you will get the default create-react-app homepage.

To access the application on any device you have connected on the same network, you can access it via http://<your-host-machine-ip>:3000 which will be shown to you in terminal.

## Project setup

The next step is to modify the App.js file located in the src folder to look like this.

*// src/App.js*

import React, {Component} from 'react';

class App extends Component {

render () {

return (

*// JSX to render goes here...*

);

}

}

export default App;

If you noticed, we got rid of import logo from './logo.svg'; and import './App.css'; along with the JSX in the render() method because we will not be needing those in the course of this tutorial.

### Setting up bootstrap

Next, we have to link bootstrap's CDN in the index.html file which can be found in the public folder.

// public/index.html

...

<head>

...

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

...

</head>

...

When this is done we will render a bootstrap card in the App.js file by including this snippet in the return() method.

*// src/App.js*

import React, { Component } from 'react';

class App extends Component {

render() {

return (

<div class="card">

<div class="card-body">

<h5 class="card-title">Steve Jobs</h5>

<h6 class="card-subtitle mb-2 text-muted">steve@apple.com</h6>

<p class="card-text">Stay Hungry, Stay Foolish</p>

</div>

</div>

);

}

}

export default App;

If we reload our application the following changes will reflect showing the contact's name, email and catch phrase in a bootstrap card.

### Feeding dynamic data from the API

Most modern web applications make use of the [REST Protocol](https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer) to communicate with each other. To achieve this, data is sent as [JSON (JavaScript Object Notation)](https://www.json.org/) to the API. In turn, the API returns a JSON payload which can be static or dynamic data. Our application will parse and display the data in a presentable manner, making it appealing to the user.

[https://jsonplaceholder.typicode.com/users](https://jsonplaceholder.typicode.com/posts)

### Creating a state

You will need npm installed, and a basic knowledge of React.

This brief tutorial will help you understand a few concepts you need to know so as to integrate a RESTful API in a React application.

React is the one of the most popular frontend framework out there and more developers are increasing learning how to build real life applications with React. While learning React, you will eventually get to a point when you need to integrate APIs in your React application.

We will be building a simple contact list application to display the contact's name, email and we will store a catch phrase of a contact. We will make use of this [endpoint](http://jsonplaceholder.typicode.com/users), it contains a JSON dump of data needed for our contact list application.

### Feeding dynamic data from the API

Most modern web applications make use of the [REST Protocol](https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer) to communicate with each other. To achieve this, data is sent as [JSON (JavaScript Object Notation)](https://www.json.org/) to the API. In turn, the API returns a JSON payload which can be static or dynamic data. Our application will parse and display the data in a presentable manner, making it appealing to the user.

Below is a typical JSON response from the contact list [dump](http://jsonplaceholder.typicode.com/users):

[

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

}

]

### Creating a state

A state is simply an object that holds data pending to be rendered. This is where we will store the output from the API call.

**state = {**

**contacts: []**

**}**

We have created a state to store the output from our API request.

### Calling the API

To fetch our contact list, we will use a componentDidMount() method in our App.js file. This method is executed immediately our component is mounted and we will also make our API request in that method.

**componentDidMount() {**

**fetch('http://jsonplaceholder.typicode.com/users')**

**.then(res => res.json())**

**.then((data) => {**

**this.setState({ contacts: data })**

**})**

**.catch(console.log)**

**}**

fetch('http://jsonplaceholder.typicode.com/users') will make a GET request to the endpoint .then(res => res.json()) parses the output to JSON, .then((data) => {this.setState({ contacts: data })}) sets the value of our state to the output from the API call and finally .catch(console.log) logs any error we get to the console.

# What is a Promise?

A promise is an object that may produce a single value some time in the future: either a resolved value, or a reason that it’s not resolved (e.g., a network error occurred). A promise may be in one of 3 possible states: fulfilled, rejected, or pending. Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.

Promises are eager, meaning that a promise will start doing whatever task you give it as soon as the promise constructor is invoked.

To catch the value on success, we'll use the then() function available on the Promise instance object. The then() function is called with whatever the return value is of the promise itself.

Like synchronous **code, chaining will result in a sequence that runs in serial. In other words, you can do:**

**fetch(url)  
 .then(process)  
 .then(save)  
 .catch(handleErrors)  
;**

### Creating our contacts component

import React from 'react'

const Contacts = ({ contacts }) => {

return (

<div>

<center><h1>Contact List</h1></center>

{contacts.map((contact) => (

<div class="card">

<div class="card-body">

<h5 class="card-title">{contact.name}</h5>

<h6 class="card-subtitle mb-2 text-muted">{contact.email}</h6>

<p class="card-text">{contact.company.catchPhrase}</p>

</div>

</div>

))}

</div>

)

};

export default Contacts

The Contacts method accepts the contacts state we created earlier and then returns a mapped version of the state, which loops over the bootstrap card to insert the contact's name, email and catch phrase.

### Rendering the contacts component

Then in our render method we have to clear out whatever we had there before and pass our component along with the contacts state in there for it to be rendered.

import React, { Component } from 'react'

import Contacts from './Contacts'

class App extends Component {

...

render() {

return (

<Contacts contacts={this.state.contacts} />

)

}

}

export default App

Summary

 We built a simple contact list web page, that displayed contact details. We called a RESTful Api, and we displayed the response to the browser using React. We also covered how the componentDidMount() method works, how state works, how components works and how to fetch data from an API and parsing the data to a component.