My Udemy Bestseller Course - Building Real-Time REST APIs with Spring Boot and Deploy on AWS Cloud (Production)

React JS (React Hooks) + Spring Boot Tutorial

author: Ramesh Fadatare

react hooks reactjs spring boot

← Back to Spring Boot Tutorial

In this tutorial, we will learn how to build a simple Full Stack application using Spring boot as backend and React JS (React Hooks) as frontend.

The objective of this tutorial:

- How to use React Hooks in React application
- How to integrate React app with Spring boot application.

React is used to build user interfaces (UI) on the front end.

Spring boot is popular to develop RESTful web services and microservices.

React Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

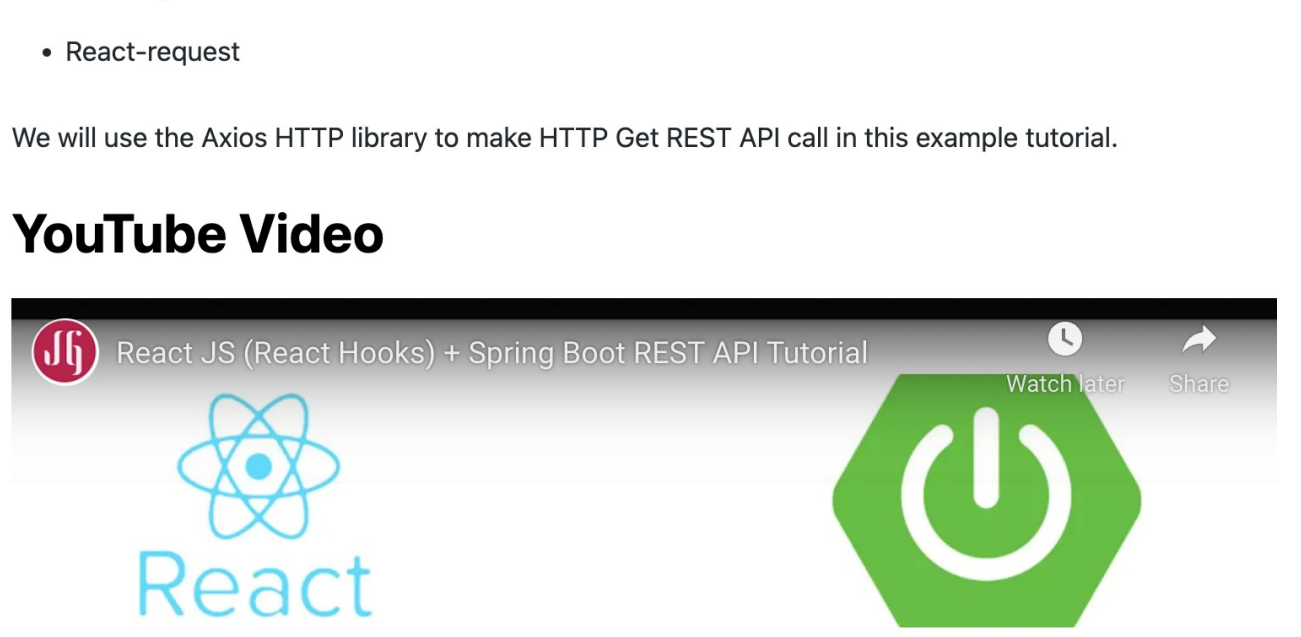
As we know, React is a JavaScript-based library that does not have the ability to make HTTP requests; thus, we need to use third-party libraries to achieve this.

There are plenty of libraries available to make HTTP calls into React apps. A few of them are listed below.

- Axios
- Fetch
- Superagent
- React-axios
- Use-http
- React-request

We will use the Axios HTTP library to make HTTP Get REST API call in this example tutorial.

YouTube Video


React JS (React Hooks) + Spring Boot REST API Tutorial
Watch later
Share
React
React (React Hooks) Frontend
Spring Boot Backend

What we will build?

We will build two projects:

1. **springboot-backend** - Develop and expose REST APIs
2. **react-frontend** - Consume REST APIs

1. Develop Spring boot application - Backend

Let's begin with creating a Spring boot application and build a simple REST API.

1. Create a Spring Boot Application

There are many ways to create a Spring Boot application. You can refer to the below articles to create a Spring Boot application.

>> **Create Spring Boot Project With Spring Initializer**

>> **Create Spring Boot Project in Spring Tool Suite [STS]**

2. Add maven dependencies

Here is a complete `pom.xml` file for your reference:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.x
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.5.3</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguides</groupId>
        <artifactId>springboot-backend</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>springboot-backend</name>
        <description>Spring Boot backend application</description>
        <properties>
                <java.version>16</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>com.h2database</groupId>
                        <artifactId>h2</artifactId>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                                <configuration>
                                        <excludes>
                                                <exclude>
                                                        <groupId>org.projectlombok</groupId>
                                                        <artifactId>lombok</artifactId>
                                                </exclude>
                                        </excludes>
                                </configuration>
                        </plugin>
                </plugins>
        </build>

</project>
```

3. Create JPA Entity - Employee.java

Let's create a new package called `entity` inside `net.javaguides.springboot` package and then create the `Employee` class inside the `entity` package with the following contents -

```java
package net.javaguides.springboot.entity;

import lombok.*;

import javax.persistence.*;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "employees")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name")
    private String lastName;
    private String email;

}
```

4. Create Spring Data JPA Repository - EmployeeRepository.java

No, we gonna create a Spring Data JPA repository to talk with the H2 database.

Let's create a new package called `repository` inside `net.javaguides.springboot` package and then create the following `EmployeeRepository` interface inside the r `epository` package -

```java
package net.javaguides.springboot.repository;

import net.javaguides.springboot.entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

5. Spring Controller with REST API - /api/employees

Let's create a new package called `controller` inside `net.javaguides.springboot` package and then create the following `EmployeeController` class inside the `controller` package with following contents -

```java
package net.javaguides.springboot.controller;

import net.javaguides.springboot.entity.Employee;
import net.javaguides.springboot.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
@CrossOrigin("http://localhost:3000/")
public class EmployeeController {

    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping("/employees")
    public List<Employee> fetchEmployees(){
        return employeeRepository.findAll();
    }
}
```

Note that we have added below line of code to avoid CORS issues:

```java
@CrossOrigin(origins = "http://localhost:3000")
```

6. Run Spring Boot Application and Test Rest API

Let's insert a few records in the `employees` table while application startup.

```java
package net.javaguides.springboot;

import net.javaguides.springboot.entity.Employee;
import net.javaguides.springboot.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringbootBackendApplication implements CommandLineRunner {

        public static void main(String[] args) {
                SpringApplication.run(SpringbootBackendApplication.class, args);
        }

        @Autowired
        private EmployeeRepository employeeRepository;

        @Override
        public void run(String... args) throws Exception {

                Employee employee1 = Employee.builder()
                                .firstName("Ramesh")
                                .lastName("Fadatare")
                                .email("ramesh@gmail.com")
                                .build();

                Employee employee2 = Employee.builder()
                                .firstName("Tony")
                                .lastName("Stark")
                                .email("tony@gmail.com")
                                .build();

                Employee employee3 = Employee.builder()
                                .firstName("John")
                                .lastName("Cena")
                                .email("cena@gmail.com")
                                .build();

                employeeRepository.save(employee1);
                employeeRepository.save(employee2);
                employeeRepository.save(employee3);
        }
}
```

Let's run this spring boot application from IDE -> Right-click -> Run As -> Java Application:

Hit this "**http://localhost:8080/api/employees**" link in a browser will popular list of employees as JSON:

```json
[
   {
      "id":1,
      "firstName":"Ramesh",
      "lastName":"Fadatare",
      "email":"ramesh@gmail.com"
   },
   {
      "id":2,
      "firstName":"Tony",
      "lastName":"Stark",
      "email":"tony@gmail.com"
   },
   {
      "id":3,
      "firstName":"John",
      "lastName":"Cena",
      "email":"cena@gmail.com"
   }
]
```

2. Build React JS Frontend Application

Let's go ahead and create a React application to consume `/api/employees` REST API.

We will use VS Code IDE to build React application.

We will use React Hook (useState) to use state in the functional components.

1 - Create a React UI with Create React App

The **Create React App** CLI tool is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

To create a new React app, open terminal in VS Code and type the below command:

```
npx create-react-app react-frontend
```

Running any of these commands will create a directory called `react-frontend` inside the current folder. Inside that directory, it will generate the initial project structure and install the transitive dependencies:

```
react-frontend
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

Let's explore important files and folders of the react project.

For the project to build, these files must exist with exact filenames:

- `public/index.html` is the page template;
- `src/index.js is` the JavaScript entry point.

You can delete or rename the other files Let's quickly explore the project structure.

package.json - The `package.json` file contains all the required dependencies for our React JS project. Most importantly, you can check the current version of React that you are using. It has all the scripts to start, build, and eject our React app.

public folder - The public folder contains `index.html` . As react is used to build a single page application, we have this single HTML file to render all our components. Basically, it's an HTML template. It has a `div` element with id as `root` and all our components are rendered in this div with `index.html` as a single page for the complete react app.

src folder- In this folder, we have all the global javascript and CSS files. All the different components that we will be building, sit here.

index.js - This is the top renderer of your react app.

node_modules - All the packages installed by NPM or Yarn will reside inside the `node_modules` folder.

App.js - The `App.js` file contains the definition of our `App` component which actually gets rendered in the browser and this is the root component.

2 - Adding Bootstrap in React Using NPM

Open a new terminal window, navigate to your project's folder, and run the following command:

```
$ npm install bootstrap --save
```

After installing the bootstrap package, you will need to import it into your React app entry file.

Open the `src/index.js` file and add the following code:

```javascript
import 'bootstrap/dist/css/bootstrap.min.css';
```

src/index.js

Here is the complete code for the `index.js` file:

```javascript
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import 'bootstrap/dist/css/bootstrap.min.css';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

3. React Service Component - REST API Call

For our API calls, we will be using `Axios` . Below is the npm command to install `Axios` .

```
npm add axios
```

Below is the `EmployeeServce.js` service implementation to make our HTTP REST call via Axios.

```javascript
import axios from 'axios';

const EMPLOYEE_API_BASE_URL = 'http://localhost:8080/api/employees';

class EmployeeService{

    getEmployees(){
        return axios.get(EMPLOYEE_API_BASE_URL);
    }
}

export default new EmployeeService();
```

Note that our backend Employee endpoint is available at **http://localhost:8080/api/employees**.

Make sure that you create an object of `EmployeeService` class export it as:

```javascript
export default new EmployeeService();
```

4. Develop a React Component

Components are the building blocks of our whole react app. They are like functions that accept inputs in terms of props, state, and outputs a UI that is rendered in the browser. They are reusable and composable.

React components can be either a function component or a class component. In this example, we are going to use the functional component with Reach Hooks.

Let's create an `EmployeeComponent.js` file and add the following code to it.

```javascript
import React, {useState, useEffect} from 'react'
import EmployeeService from '../services/EmployeeService';

function EmployeeComponent() {

    const [employees, setEmployees] = useState([])

    useEffect(() => {
        getEmployees()
    }, [])

    const getEmployees = () => {

        EmployeeService.getEmployees().then((response) => {
            setEmployees(response.data)
            console.log(response.data);
        });
    };

    return (
        <div className = "container">

            <h1 className = "text-center"> Employees List</h1>

            <table className = "table table-striped">
                <thead>
                    <tr>
                        <th> Employee Id</th>
                        <th> Employee First Name</th>
                        <th> Employee Last</th>
                        <th> Employee Email</th>
                    </tr>

                </thead>
                <tbody>
                    {
                        employees.map(
                                employee =>
                                <tr key = {employee.id}>
                                    <td> {employee.id }</td>
                                    <td> {employee.firstName }</td>
                                    <td> {employee.lastName }</td>
                                    <td> {employee.email }</td>

                                </tr>

                        )
                    }

                </tbody>


            </table>

        </div>
    )
}

export default EmployeeComponent
```

Let's understand the React Hooks used in above code snippet.

useState() hook

The `useState` is a Hook (function) that allows you to have state variables in functional components. You pass the initial state to this function and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.

```javascript
    const [employees, setEmployees] = useState([])
```

useEffect() hook

The `useEffect` Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods that are available in class components. In other words, Effects Hooks are equivalent to `componentDidMount()` , `componentDidUpdate()` , and `componentWillUnmount()` lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

- Updating the DOM,
- Fetching and consuming data from a server API,
- Setting up a subscription, etc.

Example: Making REST API call:

```javascript
    useEffect(() => {
        getEmployees()
    }, [])

    const getEmployees = () => {

        EmployeeService.getEmployees().then((response) => {
            setEmployees(response.data)
            console.log(response.data);
        });
    };
```

5. App.js

In the previous step, we have created `EmployeeComponent` so let's go ahead add `EmployeeComponent` to `App` component:

```javascript
import './App.css';
import EmployeeComponent from './components/EmployeeComponent';

function App() {
  return (
    <EmployeeComponent />
  );
}

export default App;
```

6. Run React App

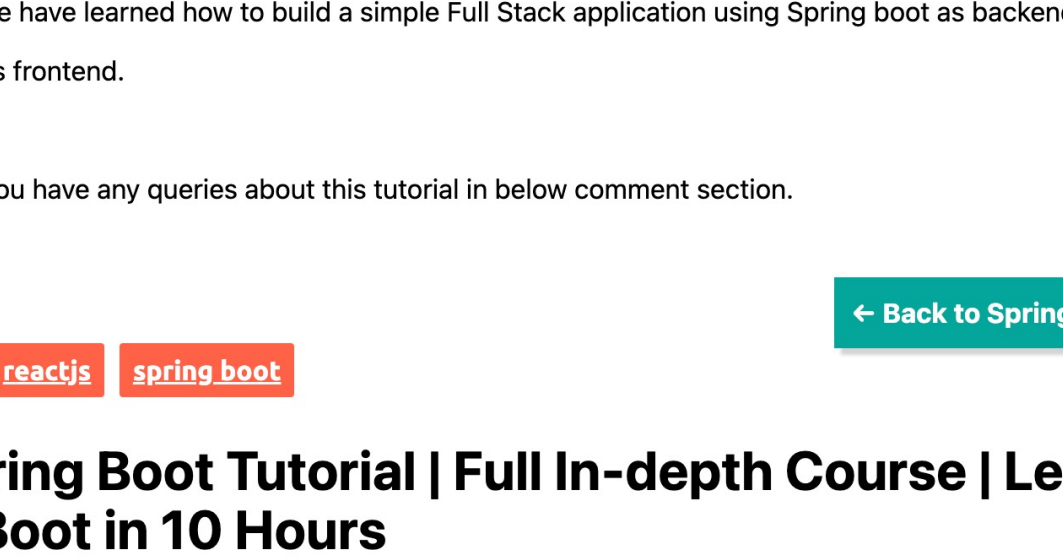
Use the below command to start the project:

```
npm start
```

Use yarn to start the project:

```
yarn start
```

Runs the app in development mode. Open **http://localhost:3000** to view it in the browser.



Employees List

Employee Id	Employee First Name	Employee Last	Employee Email
1	Ramesh	Fadatare	ramesh@gmail.com
2	Tony	Stark	tony@gmail.com
3	John	Cena	cena@gmail.com

3. Conclusion

In this tutorial, we have learned how to build a simple Full Stack application using Spring boot as backend and React JS (React Hooks) as frontend.

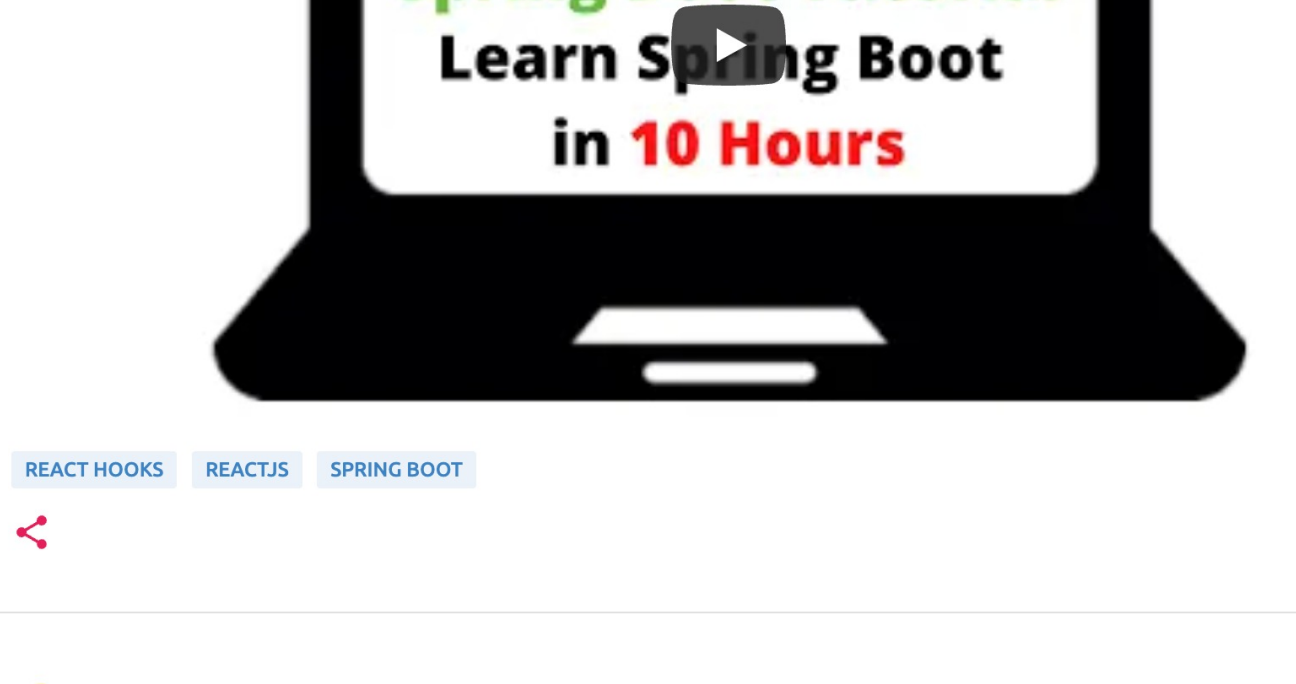
Let me know if you have any queries about this tutorial in below comment section.

← Back to Spring Boot Tutorial

react hooks reactjs spring boot

Free Spring Boot Tutorial | Full In-depth Course | Learn Spring Boot in 10 Hours

Watch this course on YouTube at **Spring Boot Tutorial | Fee 10 Hours Full Course**



REACT HOOKS REACTJS SPRING BOOT

Unknown 8 January 2022 at 01:35

cool

REPLY

To leave a comment, click the button below to sign in with Blogger.

SIGN IN WITH BLOGGER

Subscriber to my top YouTube Channel (75K+ Subscribers)
Java Guides
YouTube 77K

Testing Spring Boot Application with JUnit and Mockito

Udemy Course - Building Real-Time REST APIs with Spring Boot (Discount coupon: SPRINGBOOT_JUNE_22)

Master Spring Data JPA with Hibernate

Spring Boot + Apache Kafka - The Quickstart Practical Guide

About Me

Hi, I am Ramesh Fadatare. I am VMWare Certified Professional for Spring and Spring Boot 2022.

I am founder and author of this blog website JavaGuides, a technical blog dedicated to the Java/Java EE technologies and Full-Stack Java development.

All the articles, guides, tutorials(2000 +) written by me so connect with me if you have any questions/queries. Read more about me at About Me.

Top YouTube Channel (75K+ Subscribers): Check out my YouTube channel for free videos and courses - Java Guides YouTube Channel

My Udemy Courses - https://www.udemy.com/user/ramesh-fadatare/

Connect with me on Twitter, Facebook, LinkedIn, GitHub, andStackOverflow

Follow Me on Twitter
Follow

Facebook Likes and Shares
Like Share 19K people like this. Be the first of

Free Courses on YouTube
Java Lambda Expressions
Java Collections Framework
Java Functional Interfaces
Spring Boot Tutorial
5 Spring Boot Projects in 10 Hours
Spring Boot + Angular Full Stack
Spring Boot React Full Stack
Spring MVC Tutorial

My Udemy Courses
Building Real-Time REST APIs with Spring Boot
Testing Spring Boot Application with JUnit and Mockito
Learn Spring Data JPA with Hibernate: The Masterclass
Learn Spring Boot: The Masterclass

Connect
YouTube
Twitter
Facebook
GitHub
Linkedin
StackOverflow

Dev Tools
JSON Formatter | Beautifier
Online HTML Editor and Compiler
Base64 Encode Online
Base64 Decode Online
URL Encoder Online
URL Decoder Online

Copyright © 2018 - 2022 Java Guides All rights reversed | Privacy Policy | Contact | About Me | YouTube | GitHub

Powered by Blogger