Introducción a React-JS

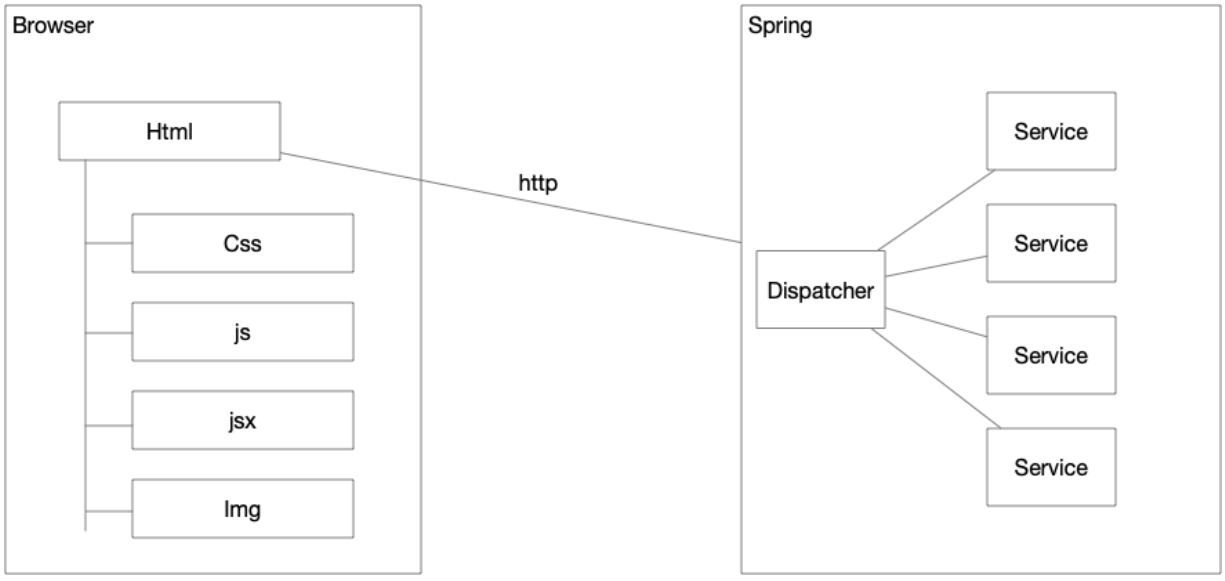
Por: Luis Daniel Benavides
Fecha: 22-06-2020
Revisado por: Rodrigo
Gualtero
Fecha: 26-09-2025

En este tutorial aprenderemos a utilizar ReactJs.

A diferencia de los miles de las guías que encontrarán internet, está guía introduce react desde un punto de vista de Arquitectura.

En la guía utilizaremos Spring para implementar servicios Web. Nuevamente esta decisión de usar Spring es para separar claramente la diferencia entre el código que se realiza en el cliente y el que se realiza en el servidor. Los ejemplos de esta guía los podría ejecutar con cualquier framework que le permita implementar servicios web, por ejemplo con node.js.

La arquitectura de la aplicación



Creando su ambiente de trabajo

Para crear su ambiente de trabajo vamos a utilizar un controlador simple de Spring que nos garantice que suba el servidor web y que empiece a servir código estático.

Para esto debe:

1. Crear una aplicación java básica usando maven.

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4
```

2. Actualizar el pom para utilizar la configuración web-MVC de spring boot. Incluya lo siguiente en su pom.

```
   <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.3.1.RELEASE</version>
        </dependency>
    </dependencies>
```

3. Cree la siguiente clase que iniciará el servidor de aplicaciones de Spring con la configuración mínima Web-MVC.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@SpringBootApplication
@RestController
public class WebSiteController {
    public static void main(String[] args) {
        SpringApplication.run(WebSiteController.class, args);
    }
    @GetMapping("/status")
    public String status() {
        return "{\"status\":\"Greetings from Spring Boot. " +
                java.time.LocalDate.now() + ", " +
                java.time.LocalTime.now() +
                ". " + "The server is Runnig!\"}";
    }
}
```

4. Cree un index html en la siguiente localización: /src/main/resources/static
5. Corra la clase que acabamos de crear y su servidor debe iniciar la ejecución
6. Verifique que se esté ejecutando accediendo a:

```
localhost:8080/status
```

7. Verifique que el servidor esté entregando elementos estáticos web entrando a:

```
    localhost/index.html
```

Nota: Spring una vez arranca los servicios Web empieza a servir recursos estáticos web que se encuentran en:

- /META-INF/resources/
- /resources/
- /static/
- /public/

Nota 2: Usted puede cambiar estos componentes estáticos de manera dinámica y el servidor los actualizará sin necesidad de reiniciarlos.

ReactJs

Vamos crear nuestra primer aplicación con ReactJS. ReactJs es un librería javascript que me permite crear clientes web pesados, dinámicos y flexibles. Es decir es una alternativa al estilo tradicional de aplicaciones web donde un servidor calcula la respuesta y me envía una página html para cada petición.

En cambio ReacJS, crea una aplicación completa corriendo en el browser que interactúa de forma dinámica con el servidor web, pero que no tiene necesariamente que cargar una página ante cada petición.

En React el programador adicionalmente puede migrar sitios web tradicionales hacia aplicaciones de una sola página por ejemplo, es decir del estado Gmail, Facebook, o twitter.

Es mucho mejor alternativa que Angular, ya que desde el punto de vista de arquitectura simplemente propone una librería de JS. No altera el modelo de funcionamiento normal del browser y no propone un lenguaje nativo, implementado con meta-información, como lo hace Angular.

Al aprender React usted aprenderá conceptos avanzados de JS.

JSX

JSX es una extensión de sintaxis a JS. JSX permite embeber código HTML dentro de componentes Js.

```
const hellomsg = <h1>Hello, world!</h1>;
```

Esto no genera una cadena o HTML, en cambio genera un elemento de React. Más de esto adelante.

Mi primer ejemplo React

Vamos a crear la primera aplicación de React. Para esto vamos a modificar nuestro entorno agregando lo siguiente al index.html antes del tag de cierre del cuerpo (</body>):

```
<!-- Load React. -->
<!-- Note: when deploying, replace "development.js" with "production.min.js". -->
<script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin>
</script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
crossorigin></script>
```

Para poder usar JSX debe cargar babel, un preprocessor que es una simple librería java script. Además debe anotar cada librería JSFX con la propiedad `type="text/babel”`. Antes de poner en producción debe compilarlas y arreglar las referencias correspondientes. El siguiente código muestra cómo debe incluir babel y la primera librería propia `"js/FirstComponent.jsx”`.

```
<!-- Load babel to translate JSX to js. -->
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>


<!-- Load our React component. -->
<script src="js/FirstComponent.jsx" type="text/babel"></script>
```

Ahora vamos crear el primer componente. Para esto cree el archivo `“FirstComponent.jsx”` en el directorio `“src/main/resources/static/js”` con le siguiente código:

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

Este código actualiza (renderiza) el elemento `“root”` del DOM virtual de React con el contenido de un elemento React creado a partir de la notación JSFX `“<h1>Hello, world!</h1>”`.

Por supuesto para que esto funcione en su archivo `index.html` debe incluir un elemento `“root”`:

```
<div id=“root”></div>
```

Ahora puede ejecutar su aplicación y en la url: localhost:8080/index.html podrá ver algo como esto:

Actualizando elementos del DOM

Para actualizar un elemento DOM el proceso hasta ahora es crear el elemento e invocar el método render del DOM así:

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Sin embargo los elementos de React son inmutables. Es decir una vez usted crea un elemento usted no puede cambiar sus hijos o sus atributos.

Para tener un elemento que cambie podríamos usar el siguiente código:

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));}


setInterval(tick, 1000);
```

Este código llama cada segundo la función tick. Remplace el código del archivo FirstComponent.jsx con este código.

Generalmente se llama a ReactDOM.render solo una vez en la aplicación. Así que ahora debemos codificar de una mejor manera elementos cambiante por medio de objetos con estado.

Componentes y props

Los componentes se usan para modularizar la interfaz gráfica en partes reusables independientes.

Un componente es básicamente una función que recibe una entrada arbitraria denominada props y retorna elementos React que le dicen que debe aparecer en la pantalla.

El siguiente ejemplo muestra un componente válido para react

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Este es un componente función que retorna un componente válido.

Nota: observe que entre las llaves usted puede escribir código javascript y combinarlo así con JSX.

Usted puede usar también clases para definir componentes:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Puede seguir el tutorial de React en el siguiente link: https://reactjs.org/docs/hello-world.html

Un componente realmente avanzado

El siguiente componente es un ejemplo de un componente avanzado que se conecta a un servidor cada 5 segundos para atraer una respuesta.

```
class StatusComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      status: ""
    };
  }


  componentDidMount() {
    this.timerID = setInterval(
      () => this.checkStatus(),
      5000
    );
  }

  checkStatus(){
      fetch("/status")
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            status: result.status
          });
        },
        // Note: it's important to handle errors here
        // instead of a catch() block so that we don't swallow
        // exceptions from actual bugs in components.
        (error) => {
          this.setState({
            isLoaded: true,
            error
          });
```

```
        }
      )
  }


  render() {
    const { error, isLoaded, status } = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
         <div>
        <h1>The server status is:</h1>
        <p>
          {status}
        </p>
        </div>
      );
    }
  }
}
```

Y puede llamarlo así:

```
ReactDOM.render(
        <StatusComponent />,
        document.getElementById('status')
        );
```

Con estos elementos ya puede hacer pequeñas aplicaciones interactivas!!

Intente construir un juego interactivo usando P5.js. Intente modificar el ejemplos que se encuentran aquí:

https://ldbn.is.escuelaing.edu.co/ipp5/public_html/

Puede encontrar el código aquí:

https://github.com/dnielben/ipp5

Controller con manejo de sesión

Maneja sesiones con @Resource para no estar tan acoplado con Spring.

```
import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;



@SpringBootApplication
@RestController
public class WebSiteController {



    @Resource
    private HttpServletRequest request;



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



    @GetMapping("/status")
    public String status() {
        sessionManagement();
        String name = (String) request.getSession().getAttribute("name");
        return "{\"status\":\"Greetings from Spring Boot "
                + name + ". "
```

```
                + java.time.LocalDate.now() + ", "
                + java.time.LocalTime.now()
                + ". " + "The server is Runnig!\"}";
    }



    @GetMapping("/setname")
    public String setName(@RequestParam(value = "name", defaultValue = "Anónimo")
String name) {
        request.getSession().setAttribute("name", name);
        return String.format("Hello %s!", name);
    }



    public void sessionManagement() {
        System.out.println(request.getSession(true).getId());
    }



}
```

Ejercicio propuesto

Antes de Realizar este taller le recomendamos seguir el tutorial de ReactJs que se encuentra en:

https://reactjs.org/docs/hello-world.html

Enunciado

Revise y entienda el proyecto de introducción a la programación usando p5.js que se encuentra en

https://ldbn.is.escuelaing.edu.co/ipp5/public_html/

Puede encontrar el código aquí:

https://github.com/dnielben/ipp5

Ahora intente desarrollar un tablero interactivo que permita a múltiples usuarios dibujar en un tablero compartido. Vea este ejemplo para inspirarse:

https://ldbn.is.escuelaing.edu.co/ipp5/public_html/04eventos/index.html

El tablero debe permitir a múltiples usuarios dibujar en línea y proveer un botón de borrado. Lo que cada persona dibuje debe aparecer en el tablero de todas las otras personas. Cada persona debe iniciar con un color diferente. Cuándo alguien oprime el botón de borrar el tablero se borra para todas las personas.

AYUda: Puede utilizar llamados temporizados para simular la interacción en tiempo real.

El siguiente código muestra una página que permite a un usuario dibujar en un tableros usando P5.js

```
<html>
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/p5.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/addons/p5.dom.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/addons/p5.sound.min.js">
</script>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <script src="js/sketch.js"></script>
  </body>
```

```
</html>
```

El siguiente es el script que gerencia el código anterior.

```
//Este código asume que las librerías de P5.js ya están cargadas.

//Esta función se ejecuta una sola vez al inicio del script.
function setup() {
  createCanvas(640, 480);
}


// Esta función se ejecuta repetidas veces indefinidamente.
function draw() {
  if (mouseIsPressed === true) {
    fill(0,0,0);
    ellipse(mouseX, mouseY, 20, 20);
  }
  if (mouseIsPressed === false) {
    fill(255,255,255);
  }

}
```