CC3S2 Lab 06:

Generación de páginas con ReactJS

Setup

Usted ya debe tener instalado Node.js y npm el paquete de administrador de su sistema.

Cree un directorio `lab06` y extraiga el contenido del archivo proporcionado en el directorio. El archivo zip contiene los archivos iniciales para esta tarea.

Esta asignación requiere muchos módulos de node que contienen las herramientas necesarias (por ejemplo, Webpack (https://webpack.js.org/), Babel (https://babeljs.io/), ESLint (https://eslint.org/)) necesarios para construir una aplicación web ReactJS (https://reactjs.org/), así como un servidor web Node.js simple (ExpressJS (http://expressjs.com/)). Estos módulos se instalan al ejecutar el siguiente comando en el directorio `lab06` :

```
npm install
```

El comando recuperara alrededor de 600 módulos node utilizando alrededor de 110 megabytes de espacio en el subdirectorio `node_modules` .

Nosotros podemos usar `npm` para ejecutar las diversas herramientas que ha recuperado. Como se puede ser visto en la propiedad `"scripts"` del archivo `package.json` , los siguientes comandos de ejecución están disponibles:

- `npm run lint` : ejecuta ESLint en todos los archivos JavaScript del proyecto . El código que envíe debe ejecutar ESLint sin warnings.
- `npm run build` - Ejecuta webpack (https://webpack.js.org/) utilizando el archivo de configuración `webpack.config.js` Para empaquetar todos archivos JSX de su proyecto en un solo paquete JavaScript en el directorio `compiled` .
- `npm run build:w` - Ejecuta webpack (https://webpack.js.org/) como el comando `run build` excepto que invoca webpack con --watch (https://webpack.js.org/api/cli/#watch-options) para que supervise los componentes de React y regenere el paquete si alguno de ellos cambia. Esta opción es útil para el desarrollo, de modo que los cambios realizados a los componentes puedan ser recuperados simplemente refrescando el navegador para cargar el paquete recién actualizado. De lo contrario, debe recordar ejecutar "npm run build" después de cada cambio.

Las soluciones para todas las tareas deben ser implementadas en el directorio `lab06`.

Este Lab. utiliza ReactJS (https://reactjs.org/) , un framework popular para crear aplicaciones web . El objetivo de la tarea es familiarizarse con ReactJS para que pueda de construir la aplicación Web de su proyecto.

Para utilizar nuestra aplicación Web a través del protocolo HTTP, utilizamos un servidor web simple de Node.js . Este servidor que puede ser iniciado con el siguiente comando en el directorio `lab06` :

```
node webServer.js
```

Todos los archivos del Lab06 se pueden recuperar utilizando una URL que comience con

```
http://localhost:3000
```

Nosotros recomendamos que configure que el entorno de desarrollo para ejecutar webpack en modo watch, de modo que tendrá que ejecutar el servidor web de node y el webpack cuando construya y pruebe su proyecto. Usted podría hacer esto mediante la ejecución de los programas en diferentes ventanas de linea de comandos. Los errores de Sintaxis se detectan y reportan por Babel así que la salida de webpack es útil. En Linux el comando :

```
node webServer.js & npm run build:w
```

ejecuta el servidor web en segundo plano y el webpack en primer plano, dentro de una sola ventana.

Preparativos

En este proyecto que requiere que se utilice el patrón modelo vista controlador (MVN). Hay muchas maneras de organizar el código bajo este patrón por lo que ofrecemos un ejemplo que demuestra algunas características básicas de ReactJS y también muestra el sistema de archivos del patrón MVN que podría seguir en sus proyectos.

Debe comenzar abriendo el ejemplo en su navegador navegando a la URL

`http://localhost:3000/getting-started.html` La página muestra ejemplos de ReactJS en acción. El HTML en `getting-started.html` proporciona un div para que ReactJS pueda dibujar la aplicación. En una etiqueta `script` se incluye el paquete de JavaScript que corresponde a la aplicación `compiled/gettingStarted.bundle.js` .

El archivo de configuración de webpack `webpack.config.js (http://localhost:3000/webpack.config.js)` especifica que este paquete se creó a partir del archivo ReactJS `gettingStarted.jsx`, que es un programa JSX que renderiza el componente ReactJS denominado `Example`, en el `div` correspondiente en `getting-started.html`

Para soportar componentes reutilizables, adoptamos una organización de archivos que co-localiza los componentes ReactJS y sus hojas de estilo CSS asociadas en un subdirectorio de un directorio llamado `componentes` . El componente `Example` se encuentra en los archivos `components/example/{Example.jsx, Example.css}`.

Usted debe estudiar los archivos invocados en el el archivo `getting-started.html (getting-started.html , gettingStarted.jsx , components/example/Example.jsx)` ya que muestra las sentencias JSX y JavaScript necesarias (junto con comentarios explicativos) para ejecutar una aplicación web de ReactJS . Usted debería utilizar este patrón de archivos y adoptar esa convención de de nombres para los otros componentes que construya.

El Modelo de datos es típicamente recuperado del servidor web , que recupera los datos de una base de datos. Para evitar tener a la configuración de una base de datos para este Laboratorio le daremos una etiqueta HTML `script` para cargar los modelos de datos directamente en el DOM del navegador desde el sistema de archivos local. Los modelos van a aparecer en el DOM bajo la propiedad de nombre `cc3s2models` . Usted va a ser capaz de acceder al modelo mediante el nombre `window.cc3s2models` en un componente de ReactJS.

Tarea1: Comprender y actualizar la vista de ejemplo.

Debe revisar y comprender la vista getting-started.html y el componente `Example`
Para demostrar su comprensión, haga lo siguiente:

1. Actualice los datos del modelo para el componente `Example` para usar su nombre en lugar de " Unknown name ". Usted debe encontrar "Unknown name" y reemplazarlo.
2. Reemplace los contenidos de la región div con la clase `motto-update` en el componente `Example` con algunas sentencias JSX que muestre su nombre y un breve lema (hasta de 20 caracteres). Al igual que el nombre del usuario, el valor inicial del lema debe venir con el modelo de datos. Usted debe incluir algo de estilo para este componente en example.css .
3. Extender lo que se hizo en el paso anterior para que permita que el usuario pueda actualizar el lema que se visualiza. El valor predeterminado del lema debe continuar recuperándose a partir del modelo de datos.

Tarea 2: Crear un nuevo componente - States

Crear una nueva vista de componente que muestre los nombres de todos los estados que contienen una determinada subcadena. Su vista debe implementar un campo de entrada que acepte una subcadena. La vista se mostrará en orden alfabético para una **lista** de todos los estados cuyos nombres contienen la subcadena dada (ignorando las diferencias mayúscula - minúscula). Por ejemplo, la vista para la subcadena de 'al' debe enumerar los estados de Alabama, Alaska, y California. La página también debe mostrar la subcadena que se utiliza para filtrar los estados. Si no hay ningún estado coincidente entonces, la página web debe mostrar un mensaje apropiado (en lugar de simplemente no mostrar nada). Todos los estados deben ser mostrados cuando la subcadena esta vacía .

Como en la Tarea n° 1, le proporcionamos los datos del modelo con estados. Se puede acceder al modelo a través de `window.cc3s2models.states` después de que se incluya con:

```
<script src="modelData/states.js"></script>
```

Consulte states.js para obtener una descripción del formato de la data de los estados.

Para ayudarle a comenzar y guiarle con las convenciones del nombrado de archivos que queremos que utilice hemos proporcionado un archivo p2.html que va a cargar y mostrar el paquete `compiled/p2.bundle.js` que se genera por webpack a partir de p2.jsx que muestra el componente React `States` . Usted puede abrir este archivo en su navegador a través de la URL (`http://localhost:3000/p2.html`) . Los archivos que necesitará implementar son:
`components/states/States.jsx` : el componente ReactJS de su componente `states`.
`components/states/States.css` : cualquier estilo CSS que necesite su componente. **Debe incluir algunos estilos para su lista de estados aquí.**

Tarea 3: personalizando el Layout

Crear un ReactJS componente llamado `Header` que mostrará una cabecera personalizada en la parte superior de la vista. Agregue este encabezado a todas las aplicaciones web ReactJS en su Laboratorio (`gettingStarted.jsx` , `p2.jsx`). Note que usted **no debe** sustituir a la sección de la parte 1 (su nombre y lema). Esa sección debe estar separada de su encabezado. Use su imaginación y creatividad para crear un encabezado que sea "único". Para ello puede incluir imágenes adicionales , gráficos, lo que desee . Usted puede extender los componentes JSX/JavaScript , pero no puede utilizar componentes externos ReactJS o bibliotecas JavaScript como JQuery.

Los archivos que necesitará implementar son:

- `components/header/Header.jsx` : el componente ReactJS de su componente de encabezado. Se define una class `Header` de tipo React.Component (https://reactjs.org/docs/react-component.html) .
- `components/header/Header.css` – los estilos CSS que su componente necesita. **Usted debe incluir algo de estilo para su cabecera aquí.**

Nota: `gettingStarted.jsx` debe tener una cabecera personalizada (la de la tarea 3) debajo de la sección con el lema (de la tarea 1.2) . Todas las demás vistas de las páginas deben tener su encabezado personalizado de la tarea 3.

Tarea 4: agregue el cambio dinámico de las vistas

Cree un `p4.html` y un archivo JSX correspondiente `p4.jsx` que incluya a ambos componentes (el Componente `Example` y `States`). El `p4.jsx` necesita implementar la capacidad de cambiar entre la visualización de los dos componentes. Cuando se muestra una vista, debe haber un botón encima que sirva para mostrar la otra vista. Por ejemplo, cuando se muestra la vista States, el botón de arriba debería ser "Cambiar a Example" y, cuando se pulsa, los estados deberían desaparecer y la vista de `Example` debería mostrarse.
Para este problema, deberá crear los archivos anteriores y modificar el archivo de configuración del webpack `webpack.config.js` para construir un archivo `compiled/p4.bundle.js` que pueda usar en el archivo p4.html.
Nota: si está utilizando Webpack con --watch (es decir, npm run build: w), deberá reiniciarlo después de cambiar código de `webpack.config.js`.