# Node.js: Cómo crear una aplicación web básica con React (Parte 3)

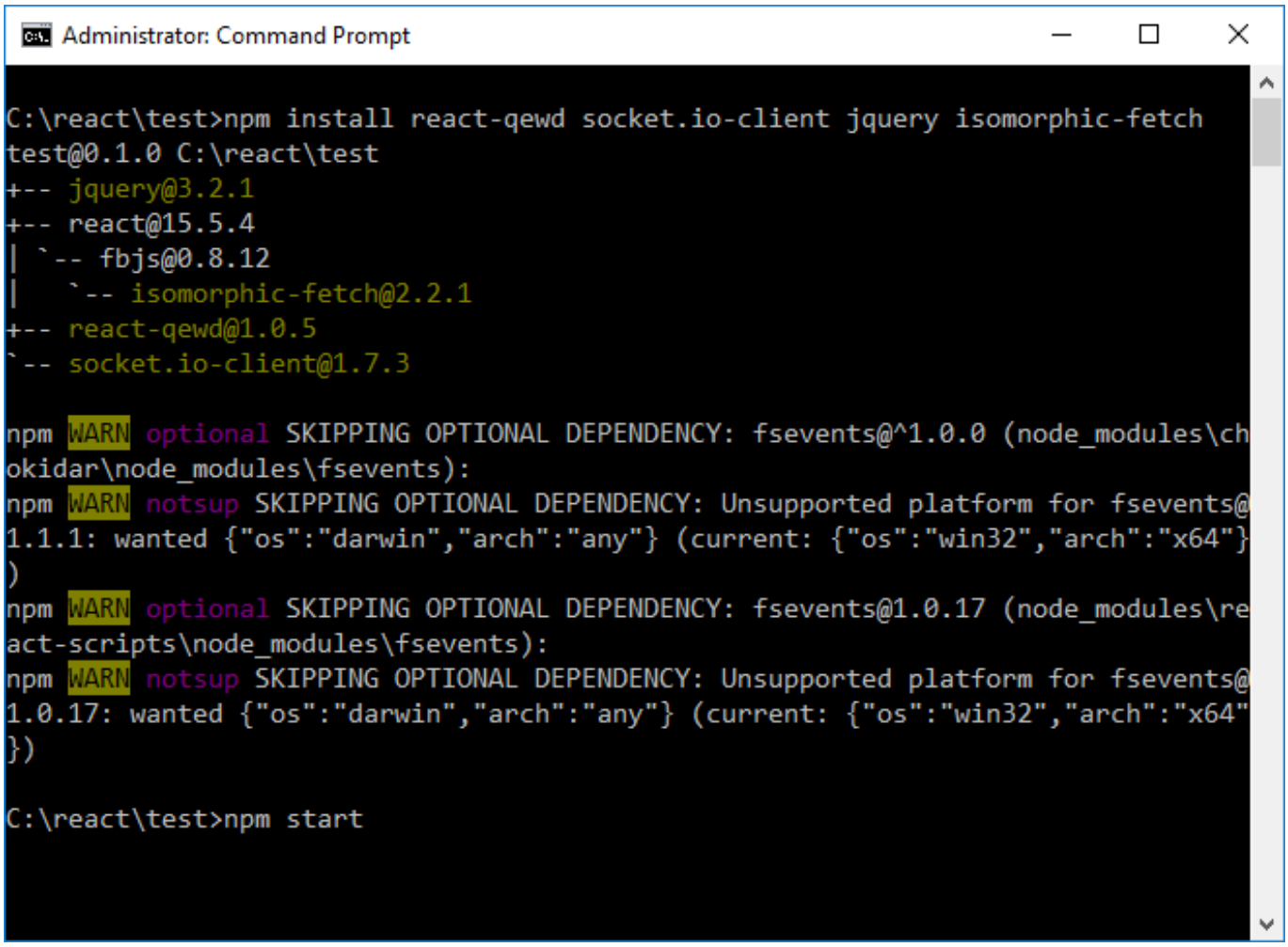
Artículo
Javier Lorenzo Mesa · Ago 5, 2020 Lectura de 13 min

# Node.js: Cómo crear una aplicación web básica con React (Parte 3)

Desarrollar una aplicación web Full-Stack en JavaScript con Caché requiere que juntes los bloques correctos para construirla. En esta tercera parte de la serie de artículos mostraré cómo vincular nuestra aplicación de React a los tres backends que creamos en la parte 2.

Si tienes la aplicación de la parte 1 ya funcionando, ahora añadiremos un poco de código JavaScript para conectarla a Caché. Asumiré que tu aplicación React aún se está ejecutando en localhost:3000 y que su servidor QEWD de la parte 2 también se está ejecutando en su propia ventana de línea de comandos en localhost:8080.

Primero, para tu aplicación React app usando Ctrl-C, instalaremos dos módulos de nodo en tu directorio C:\react\test:




De nuevo, ignora las advertencias: acabamos de instalar el paquete react-qewd, que deja el módulo ewd-client

disponible para usar en una aplicación React. Tu navegador usa el ewd-client para conectarse al servidor de backend QEWD. El contenido del módulo socket.io-client abrirá nuestro WebSocket. El ewd-client usa de forma predeterminada el módulo jquery para soportar el modo Ajax (opcional). El módulo isomorphic-fetch es necesario para hacer llamadas REST al backend QEWD/REST (opción 2) y CSP/REST (opción 3).

Ahora, cierra la instancia anterior de la aplicación React en Chrome y reinicia tu aplicación React con el comando **npm start**.

Ahora implementaremos la misma llamada de backend usando los tres backends que creamos en la parte 2. Esto te permitirá ver con claridad las diferencias en el código y qué funciona mejor para ti.

---

**Primera opción: conectar a Caché usando QEWD/WebSockets**

Ve al proyecto de prueba React en tu editor Atom y edita el archivo **index.js** file en el subdirectorio **src**:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';
import { QEWD, QEWDProvider } from 'react-qewd';
let qewd = QEWD({
  application: 'test', // application name
  log: true,
  url: 'http://localhost:8080'
});
ReactDOM.render(
  <QEWDProvider qewd={qewd}>
    <App />
  </QEWDProvider>,
  document.getElementById('root')
);
```

Añadimos el módulo react-qewd, definimos la instancia de objeto qewd y agregamos el componente de aplicación externo QEWDProvider para pasar el objeto qewd hacia abajo como una propiedad a los componentes subyacentes de React (aplicación).

A continuación, edita **App.js** en **src**:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { message: 'no message received yet!' };
  }

  handleClick = () => {
    let me = this;
    let { qewd } = this.props;
    let messageObj = {
      type: 'isctest',
      //ajax: true,
      params: {
        text: 'my ISC test message'
      }
    };
```

```
    qewd.send(messageObj, function(messageObj) {
      //console.log(messageObj);
      me.setState(prevState => ({
        message: messageObj.message.text
      }));
    });
  }

  render() {
    let { qewdProviderState } = this.props;
    return (
      <span>
        {
          qewdProviderState.registered ?
            <div className="App">
              <div className="App-header">
                <img src={logo} className="App-logo" alt="logo" />
                <h2>Welcome to React</h2>
              </div>
              <p className="App-intro">
                To get started, edit <code>src/App.js</code> and save to reload.
              </p>
              <button onClick={this.handleClick}>Send message to ISC</button>
              <p className="App-intro">
                {this.state.message}
              </p>
            </div>
          :
            <p className="App-intro">
              Registering QEWD ...
            </p>
        }
      </span>
    );
  }
}
export default App;
```

Añadimos varias piezas de código aquí:

- un constructor para el componente de aplicación donde inicializamos su estado (aún no se recibieron mensajes de Caché)
- un controlador handleClick donde el mensaje se envía a Caché a través del WebScoket. Como puedes ver, las comunicaciones entre componentes quedan totalmente ocultas por el método qewd.send()
- definimos el objeto qewdProviderState en el método render(), que el QEWDProvider pasa automáticamente
- renderizado condicional dependiendo del booleano qewdProviderState.registered: mientras el WebSocket está estableciendo su conexión al servidor QEWD en el backend, solo mostramos un párrafo HTML que diga "Registering QEWD ..." ("Registrando QEWD"). En cuanto esté lista la conectividad del WebSocket, la aplicación se volverá a renderizar a sí misma de forma automática y mostrará nuestra interfaz de usuario completa. Esta funcionalidad es la que hace tan potente a React: usa un DOM virtual para volver a renderizar tu interfaz de usuario de la forma más eficiente posible, modificando únicamente las partes de la interfaz de usuario que cambiaron
- un botón para enviar el mensaje a QEWD e invocar el controlador handleClick
- un párrafo que muestre el mensaje que vuelve de Caché

Guarda este archivo también y observa cómo tu Chrome vuelve a cargar la aplicación React. Ahora abre primero las herramientas de Chrome para desarrolladores (Devtools) con Ctrl+Mayús+I y ve a la pestaña de depuración de

React. Abre el componente QEWDProvider y haz clic en el componente de la aplicación. A la derecha deberías ver los accesorios y el estado de este componente.

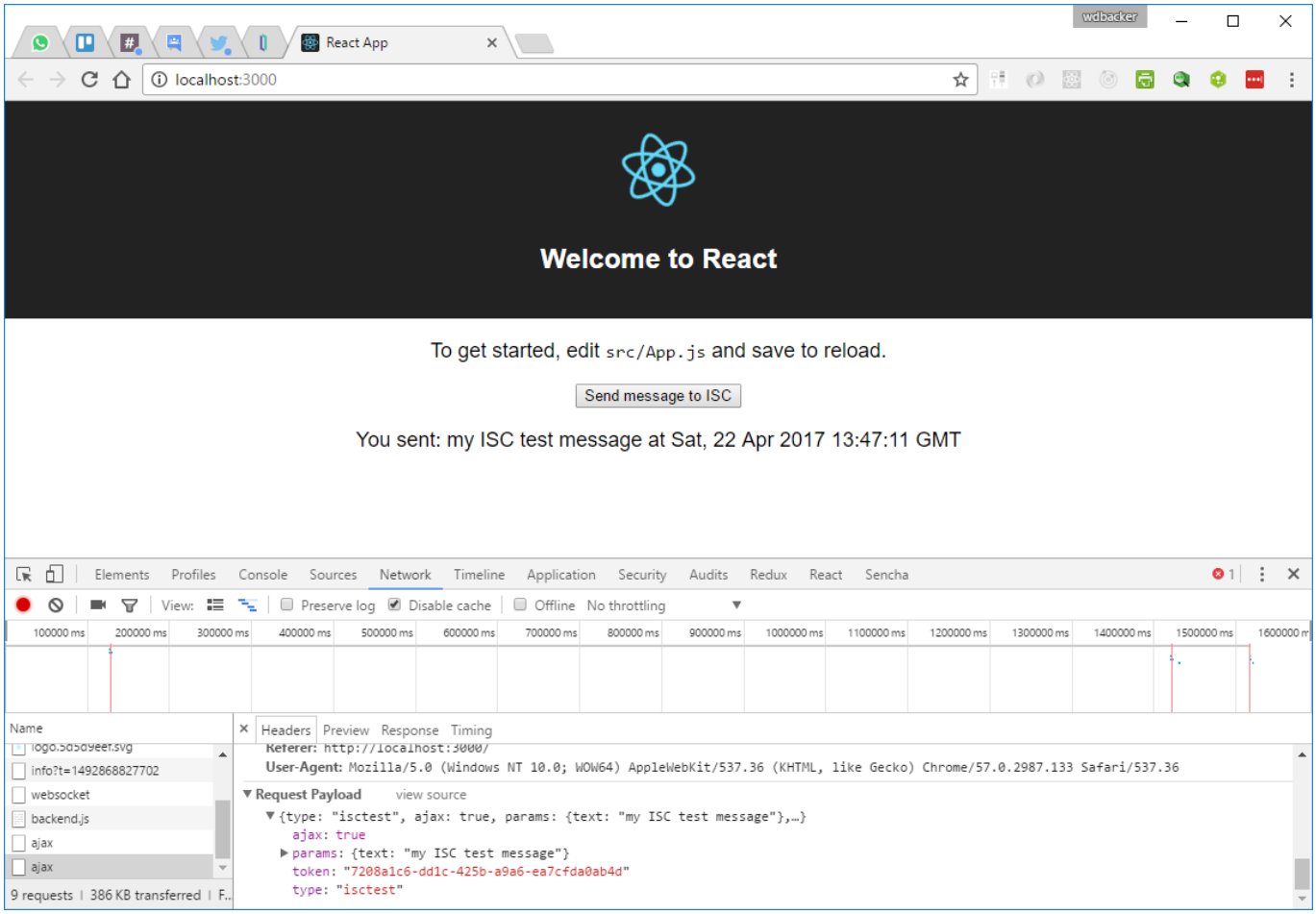
Ahora haz clic en el botón "Send message" (enviar mensaje) y observa cómo el mensaje se envía en Devtools- actualizará la propiedad del mensaje en el estado y tu primer mensaje WebSocket aparecerá también en tu HTML.

¡Enhorabuena! ¡Ya has terminado tu primera aplicación WebSocket de React con Caché en el backend!

Ahora también puedes ver los contenidos del global ^nodeTest con solo abrir un terminal de Caché. Deberías ver que el mensaje aparece con una marca temporal distinta cada vez que presionas el botón.

Por cierto, también puedes usar herramientas similares para supervisar y depurar tu código de backend que se ejecuta en QEWD. Te mostraré cómo hacer esto en un futuro artículo sobre depuración.

Puedes probar una última cosa: quita las marcas de comentario a "ajax: true" dentro del messageObj en handleClick(). Guarda Apps.js, abre la pestaña de depuración de red en las devtools de Chrome y observa cómo ahora el mismo mensaje sale a través de Ajax:




Nota: no fue necesario reescribir nuestro código, tan solo habilitamos el modo Ajax para este mensaje. Mira también la respuesta en la pestaña de depuración de respuestas.

* Por cierto, no es necesario que hagas esto en cada mensaje, también puedes activar ajax para toda la aplicación si lo activas en la configuración qewd en **index.js**. Todos los mensajes se enviarán usando Ajax en este modo.

Verás que ahora aparece un token dentro del mensaje: QEWD genera esto para cada mensaje que envíes al backend y garantiza una comunicación segura gracias a que cada sesión de cliente tiene un token único. Esto también es algo que hace muy atractivo a QEWD: se encarga de todo el trabajo "oculto", lo que le permite al

desarrollador concentrarse en el código de su aplicación.

Antes de comenzar con tus propios experimentos con esta tecnología React, te recomiendo que, antes de nada, eches un vistazo a estos tutoriales sobre React. Es un paso necesario para entender la diferencia entre HTML plano y JSX (las etiquetas "HTML" en el código). ¡HTML y JSX no funcionan igual! Estos documentos también contienen información importante acerca de cómo escribir tu JavaScript correctamente, ¡te ahorrará un montón de tiempo (de depuración)!

---

**Segunda opción: conectar a Caché usando QEWD/REST**

Ahora cambiaremos el código de esta misma aplicación para usar llamadas REST. Para reducir al mínimo las modificaciones, mantendré el código del WebSocket QEWD en **index.js** ya que no interfiere con nuestras modificaciones para llamadas REST.

Simplemente edita App.js y modificar tu código para REST (en negrita):

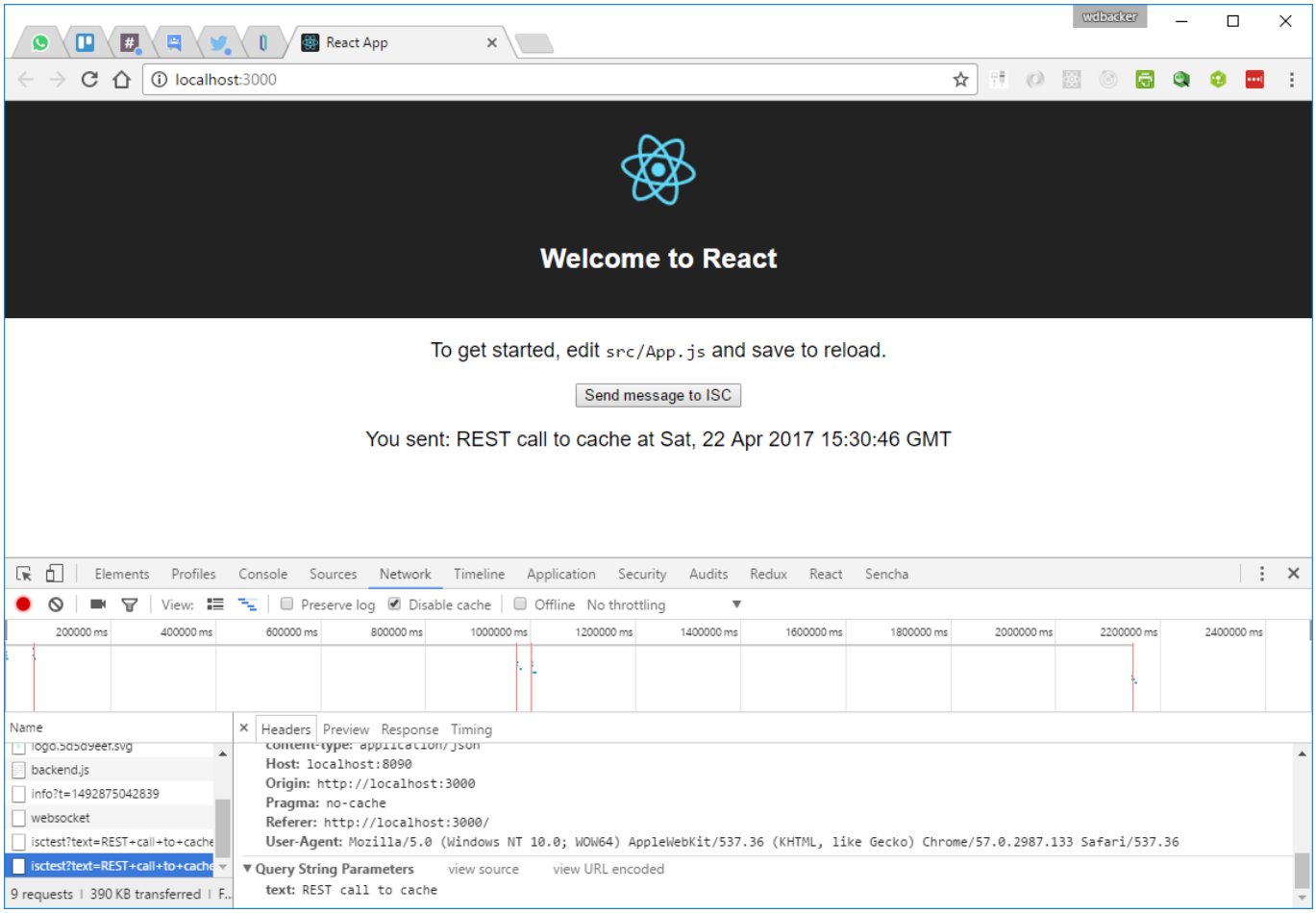
```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import fetch from 'isomorphic-fetch';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = { message: 'no message received yet!' };
  }
  handleClick = () => {
    let me = this;
    /*
    let { qewd } = this.props;
    let messageObj = {
      type: 'isctest',
      ajax: true,
      params: {
        text: 'my ISC test message'
      }
    };
    qewd.send(messageObj, function(messageObj) {
      //console.log(messageObj);
      me.setState(prevState => ({
        message: messageObj.message.text
      }));
    });
    */
    var headers = new Headers({
      "Content-Type": 'application/json'
    });
    fetch('http://localhost:8080/testrest/isctest?text=REST+call+to+cache', {
      method: 'GET',
      headers: headers,
      mode: 'cors',
      timeout: 10000
    })
    .then(response => response.json())
    .then(response => {
      me.setState(prevState => ({
        message: response.text
      }));
```

```
    })
  }
  render() {
    let { qewdProviderState } = this.props;
    return (
      <span>
        {
          qewdProviderState.registered ?
            <div className="App">
              <div className="App-header">
                <img src={logo} className="App-logo" alt="logo" />
                <h2>Welcome to React</h2>
              </div>
              <p className="App-intro">
                To get started, edit <code>src/App.js</code> and save to reload.
              </p>
              <button onClick={this.handleClick}>Send message to ISC</button>
              <p className="App-intro">
                {this.state.message}
              </p>
            </div>
          :
            <p className="App-intro">
                Registering QEWD ...
            </p>
        }
      </span>
    );
  }
}
export default App;
```

Como puedes ver, solo necesitamos marcar como comentario el código qewd.send() y sustituirlo con un fetch() estándar de JavaScript. Esta es la nueva sintaxis para emitir solicitudes Ajax en el navegador y en Node.js (de ahí el nombre del módulo: isomorphic fetch, trabaja tanto en el servidor como en el cliente).

Vuelve a guardar App.js y pruébela en Chrome presionando el botón otra vez. Ahora verás la llamada a REST en Devtools:




Como puedes ver, ¡no fue necesario cambiar mucho nuestra aplicación para usar una tecnología de backend totalmente distinta!

---

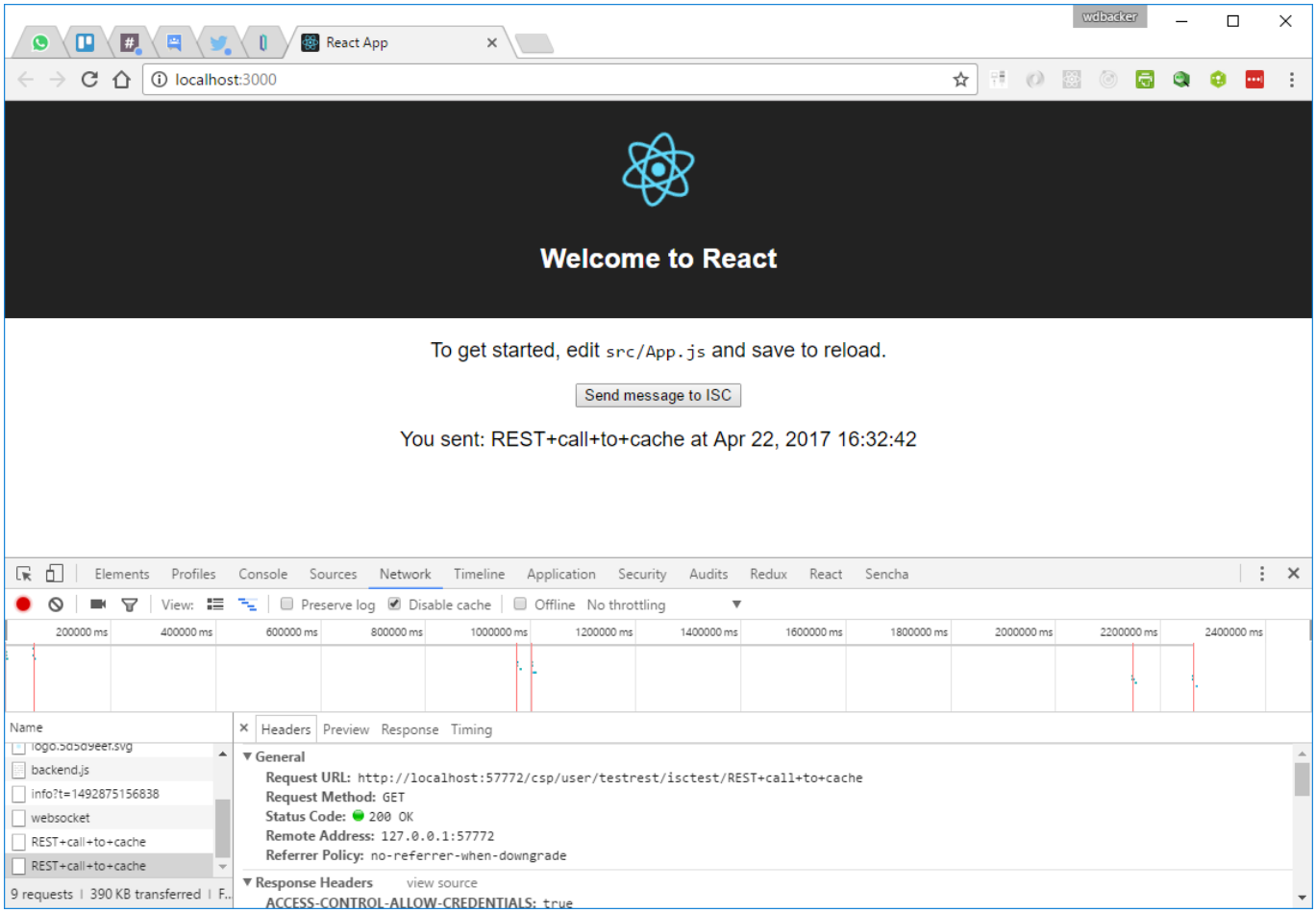
**Tercera opción: conectar a Caché usando CSP/REST**

Esta es realmente fácil de modificar: solo tenemos que modificar nuestra url REST:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import fetch from 'isomorphic-fetch';
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { message: 'no message received yet!' };
  }
  handleClick = () => {
    let me = this;
    /*
    let { qewd } = this.props;
    let messageObj = {
      type: 'isctest',
      ajax: true,
      params: {
        text: 'my ISC test message'
      }
    };
    qewd.send(messageObj, function(messageObj) {
```

```
      //console.log(messageObj);
      me.setState(prevState => ({
        message: messageObj.message.text
      }));
    });
    */
    var headers = new Headers({
      "Content-Type": 'application/json'
    });
    fetch('http://localhost:57772/csp/user/testrest/isctest/REST+call+to+cache', {
      method: 'GET',
      headers: headers,
      mode: 'cors',
      timeout: 10000
    })
    .then(response => response.json())
    .then(response => {
      console.log(response);
      me.setState(prevState => ({
        message: response.text
      }));
    })
  }
  render() {
    let { qewdProviderState } = this.props;
    return (
      <span>
        {
          qewdProviderState.registered ?
            <div className="App">
              <div className="App-header">
                <img src={logo} className="App-logo" alt="logo" />
                <h2>Welcome to React</h2>
              </div>
              <p className="App-intro">
                To get started, edit <code>src/App.js</code> and save to reload.
              </p>
              <button onClick={this.handleClick}>Send message to ISC</button>
              <p className="App-intro">
                {this.state.message}
              </p>
            </div>
          :
            <p className="App-intro">
                Registering QEWD ...
            </p>
        }
      </span>
    );
  }
}
export default App;
```

Una vez más, ahora puedes probarla fácilmente en Chrome usando CSP/REST:




¡Enhorabuena! Has creado una aplicación React con tres backends posibles.

Este ejemplo fue lo más reducido posible para mostrar los principios básicos del desarrollo moderno de aplicaciones JavaScript. Por supuesto, esto es solo el comienzo y espero que empieces a desarrollar JavaScript con Full-Stack usando Caché. ¿Quizás deberíamos llamar a esta nueva colección de tecnologías el stack CNQR (Caché/Node.js/QEWD/React)?

Ten en cuenta también que puedes sustituir fácilmente React con AngularJS, Vue.js... o con cualquier otra estructura JavaScript. Esta forma de trabajo te permite cambiar tu frontend bastante rápido sin tener que hacer demasiadas modificaciones a tu backend.

También habrás notado que la definición declarativa de interfaz de usuario usada por React en este ejemplo aún contiene la lógica empresarial (controladores de eventos) en el mismo archivo fuente. Con React, puedes hacerlo mejor y crear una separación limpia entre la definición de la interfaz de usuario y su lógica. Para opciones más potentes de gestión de datos del lado del cliente, puedes usar el módulo Redux. Ofrece un almacén central para el estado de tu aplicación (en nuestro ejemplo, el estado de la interfaz de usuario se guarda en el componente de la aplicación). No lo introduje en este ejemplo a propósito, ya que hace que el código sea mucho más grande. Pero, para aplicaciones de mayor tamaño, ¡sin duda que es una excelente forma de estructurar el código de tu aplicación! Intentaré cubrir este módulo en una parte posterior de esta serie.

#API REST #JavaScript #JSON #Node.js #React #Caché
00 2 0 0 211

Mensajes relacionados

- Node.js: Cómo crear una aplicación web básica con React (Parte 1)
- Node.js: Cómo crear una aplicación web básica con React (Parte 2)
- Node.js: Cómo crear una aplicación web básica con React (Parte 3)

Log in or sign up to continue
Añade la respuesta

**URL de fuente:** https://es.community.intersystems.com/post/nodejs-c%C3%B3mo-crear-una-aplicaci%C3%B3n-web-b%C3%A1sica-con-react-parte-3