Overview

- Use reactjs to build single-page users application using previously developed web services.
- Use axios for accessing web services; access code virtually identical to that to earlier user-ws.js except modularization changed from nodejs `module` to ES6 `class`.
- Functionality to allow creating, updating, deleting users.
- Setup to access user-ws web-services (which must be running).

Same Origin Policy

- Cookies are often used to retain authentication state; i.e. once a user has authenticated with a web site, that web site gives the browser a cookie set to a special authentication token specifying a authenticated session.
- Whenever the browser sends a request to that web site domain it will include that cookie to indicate that the user is authenticated.
- It is possible that the JavaScript loaded from a third-party web site also makes a request to the original web site. If this is permitted then the browser would send the authentication token and the third-party web site would have authenticated access to the original web site.
- To prevent such attacks, browser enforce **the same origin policy**: JavaScript loaded from one web site cannot make requests to another web site.
- Same-origin means identical protocol (i.e. `http`, `https`), domain and port.

Cross-Origin Resource Sharing

- Acronym CORS.
- To allow mashup web pages where a page includes contents from multiple browsers, the standards make it possible to relax the same-origin policy.
- A web site can whitelist other domains to access its services.
- For non-sensitive services, all domains can be whitelisted.

AJAX

- AJAX stands for **Asynchronous JavaScript and XML**.
- Allows the browser to asynchronously make responses to remote servers without any involvment by the user.
- Makes web applications possible .. **Single Page App** or SPA.
- Does not require the use of XML, often used with JSON.

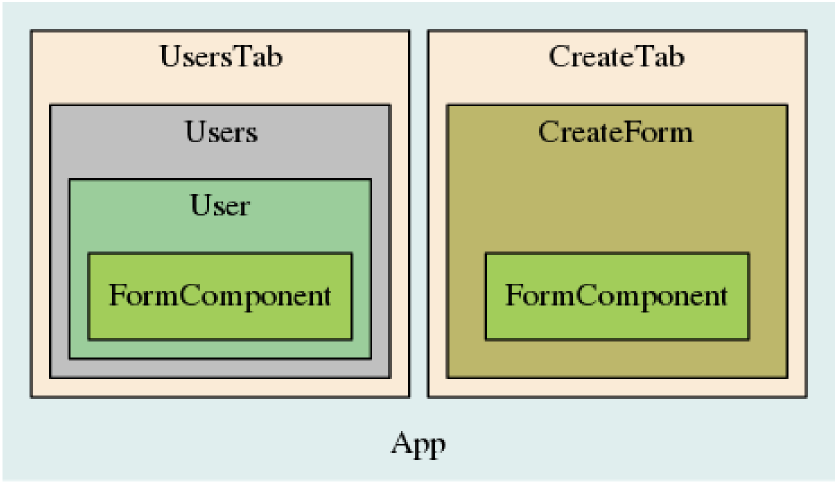
XMLHttpRequest

- Low-level API used to make asynchronous calls. Example:

```
const req = new XMLHttpRequest();
req.onreadystatechange = () => {
  if (this.readyState == 4 &&
      this.status == 200) {
    //insert into page
    document.getElementById("element").
      innerHTML = req.responseText;
  }
};
req.open("GET", someUrl);
req.send();
```

- Request states: 0: unsent, 1: opened, 2: headers received, 3: loading, 4: done.
- Application uses axios to hide this complexity.

Component Nesting



Passing Info Up and Down Component Hierarchy

- It is easy to pass information **down** the hierarchy using `props`. Trivial to provide default values for properties.
- Pass information **up** the hierarchy by using callbacks. Those callbacks are passed down the hierarchy using `props`.
- Since the `FormComponent` is generic it does not know what to do when a form is submitted; we pass down a handler which the `FormComponent` calls.
- We instantiate web-services at the top-level and pass it down the hierarchy using a `ws` property. The actual web service calls are made by the level just above the `FormComponent`.
- We also pass the app all the way down the hierarchy so that deep components can switch the tabs using `app.select()`.
- Note that reactjs has a context which can be used to hold stuff like the web services.

Calling Component Methods from Outside Component

Does not work.

```
class C extends React.Component {
  m() { ... }
}

const c = <C/>;
c.m(); //error
```

React has wrapped the object instance to build the component and the wrapped component does not have a method `m()`.

Calling Component Methods from Outside Component Continued

What works is to pass the object instance under the component and then call the method on the instance.

```
class C extends React.Component {
  m() { ... }
  h() { ... <C1 x={this}>... }
}

class C1 extends React.Component {
  constructor(props) {
    super(props);
    props.x.m(); //this should work
  }
}
```

Code

- Application setup.
- Code Directory.
- *jsx code*.
- Fiddle for pure CSS tabs.