REACT REVIEW

You should understand:

- React components as functions
- React components are composable
- Components can pass props to other components
- How to set the state of a component
- State and prop changes trigger re-renders

A BIGGER EXAMPLE

Consider:

```
// ...
  let content;
  if(userState.isLoggedIn) {
    content = <SecretStuff user={userState}/>;
  } else {
    content = <Login onLogin={ login }/>;
  }

  return (
    <div className="app">
      <Nav user={userState} onLogout={logout}/>
      {content}
    </div>
  );
// ...
```

CONCLUSIONS

Readability:

- `login`/`logout` are functions
    - *inside* the component
- `content` is just for cleanliness

Separation of Concerns:

- `userState` is managed in this component
- other components have it as a **read-only** prop

SIDE NOTE: PROPS ARE READ ONLY

Props should be read-only

A common mistake is to use them as the state and update them

This causes problems when they change in the ancestor/sibling or on a re-render

Which version is true?

Avoid this: **Props are read-only**

CONSIDERING NAV

```
import { fetchLogout } from './services';

const Nav = ({ user, onLogout }) => {
  const logout = () => {
    fetchLogout()
    .then( () => onLogout() );
  };
  return (
    <ul className='nav'>
      <li>{user.username}</li>
      { user.isLoggedIn && <li onClick={logout}>Logout</li> }
    </ul>
  );
};
```

NAV THOUGHTS?

Why an `<ul>`?

- Common menu structure, unrelated to React
- React doesn't change these conventions

`user` isn't called `userState` here

- it's not state, it's a prop
- change is made through `onLogout()`

Service call is encapsulated within `Nav.jsx`

- App doesn't know HOW Nav did the logout

Why lack of error messaging?

LOGIN

```
return (
  <div className="login">
    <p className="error">{error}</p>
    <input onChange={ (e) => setUsername(e.target.value) }/>
    { isLoading ?
        <img alt="spinner" src={spinner}/> :
        <button onClick={ performLogin }>Login</button>
    }
  </div>
);
```

BASIC LOGIN THOUGHTS

So much state **local to component**

- `isLoading`
- `error`
- `username`

Logic in JSX file, but out of JSX syntax

- `performLogin`

Spinner!

MORE LOGIN CODE

```
const performLogin = () => {
  if(!username) {
    setError(messages.USERNAME_REQUIRED);
    return;
  }
  setError('');
  setIsLoading(true);

  fetchLogin(username)
  .then( (userInfo) => onLogin(userInfo.username) })
  .catch( (err) => {
    setError(messages[err.code || 'DEFAULT']);
    setIsLoading(false);
  });
};
```

MORE LOGIN THOUGHTS

Note lack of state "cleanup"

- Assumes component will be removed after login

What is `messages`? Why CAPS?

- common CONSTANTS convention
- external config file:

```
export default messages = {
  DEFAULT: 'Oh no!  Something went wrong, please try again',
  USERNAME_REQUIRED: 'Username is required',
  NETWORK_ERROR: 'There was a problem reaching your network, please try again',
  LOGIN_REQUIRED: 'You must be logged in to view this content',
  LOGIN_UNAUTHORIZED: 'You are not permitted to view this content',
};
```

HOW TO TIE INTERACTIONS TO RENDERING

`useEffect` hook:

- does NOT return anything
- runs when component renders
- or only when any listed vars change

USEEFFECT EXAMPLE

```
useEffect( () => {
  fetchLoginStatus()
  .then( userInfo => {
    setUserState({
      isLoggedIn: true,
      username: userInfo.username,
    });
  });
}, []);
```

USEEFFECT THOUGHTS

- Here we are rendering only on **initial** render
    - the `[]` means it only runs once
    - an array of vars means rerun only if they change
    - ...no vars in array, so nothing triggers change
- If component removed and replaced, total new state
    - But that can't happen for this example

USEEFFECT ADVICE

- Use cautiously
- Load state as needed
- Remember spinners to account for delay