Debugging React Apps

Lesson 01

Capgemini

Lesson Objectives

At the end of this module you will be able to:

- Understanding React Error Messages
- Handling Logical Errors,
- Debugging React apps using google developer tools and React DevTool
- Understanding Error Boundaries

Understanding React Error Messages

One of the most important things a developer should learn is how to (properly) debug an application.

This educates you to easily find out the cause of errors in code, but also teaches internal working of the language thus preventing future errors.

Console window is one of *the* most important tools at your disposal.

It helps as to se the state of application which we generally print using Console.log

It also shows the majority of error messages in browser caused by your app.

Disadvantages of Console.log :

when used with an object/array as a parameter can be deceiving.

```
const myObj = { name: `Bala`, age:23 }
console.log(myObj);
myObj.name = 'John';
myObj.age = 22;
```

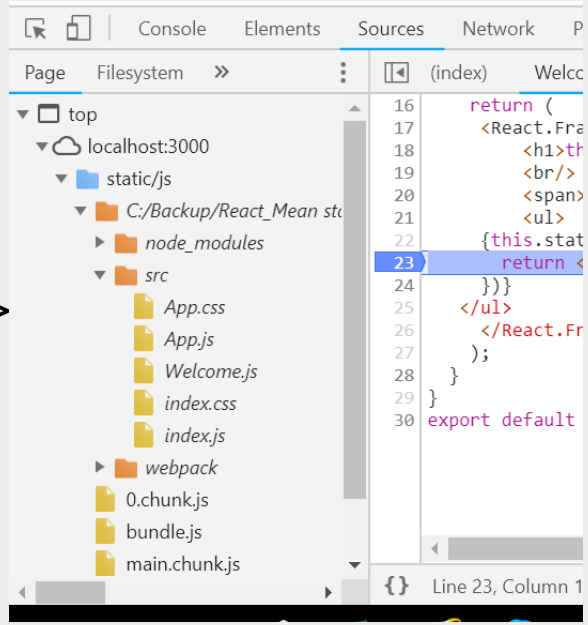
Handling Logical Errors

Logical errors are difficult to spot mostly in an application. Generally errors will be displayed in the browser screen and console, but logical errors wont.

Logical errors can be identified by using developer tools (F12).Following is the step to find out the errors.

1. Click F12 (developer tools).
2. Goto Sources tab in the developer tools
3. In the left side panel, we have a tab called page under which there is

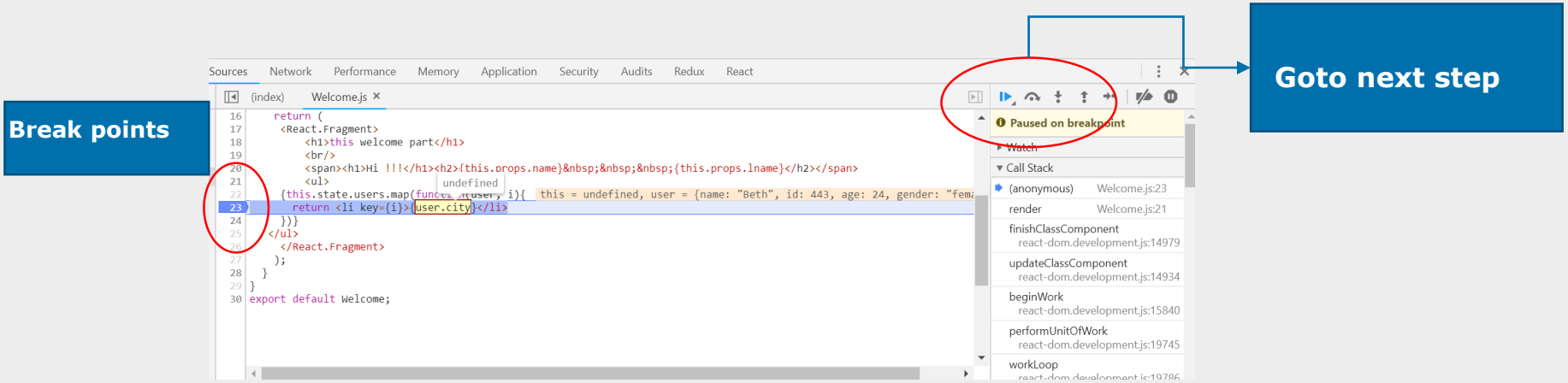
localhost:3000 -> static/js -> C:\Backup\
2019 react js\Demos_2019\react-debug-errors ->
Src -> Welcome.js
(shown in screenshot on right)



4. double click on the line numbers in the Welcome.js file as shown below in the screen shot.

5. Refresh the page once show that the debugger starts and use the buttons on right hand side for navigating between break points

We can have any number of break points. Thus we can identify the errors in the code.



Demo

React-debug-logical-errors (use dev tools F12)

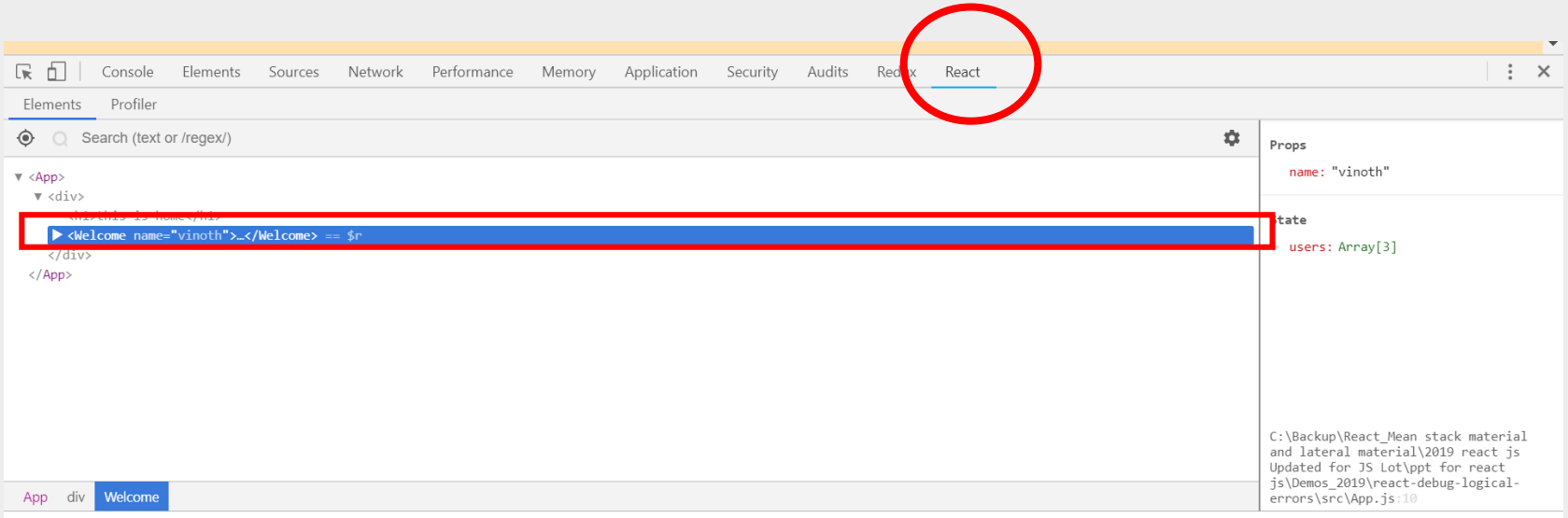
Debugging React apps using google developer tools and React DevTool

- React Developer Tools lets you inspect the React component hierarchy, including component props and state.
- It exists both as a browser extension (for Chrome and Firefox), and as a standalone app (works with other environments including Safari, IE, and React Native).
- We are using Chrome extension for debugging in demos from below link.
  - https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en
- A quick way to bring up the DevTools is to right-click on the page and press Inspect.
- This you can select ***react*** option in the developer tools tab and you select any elements we need.

Debugging React apps using google developer tools and React DevTool contd.

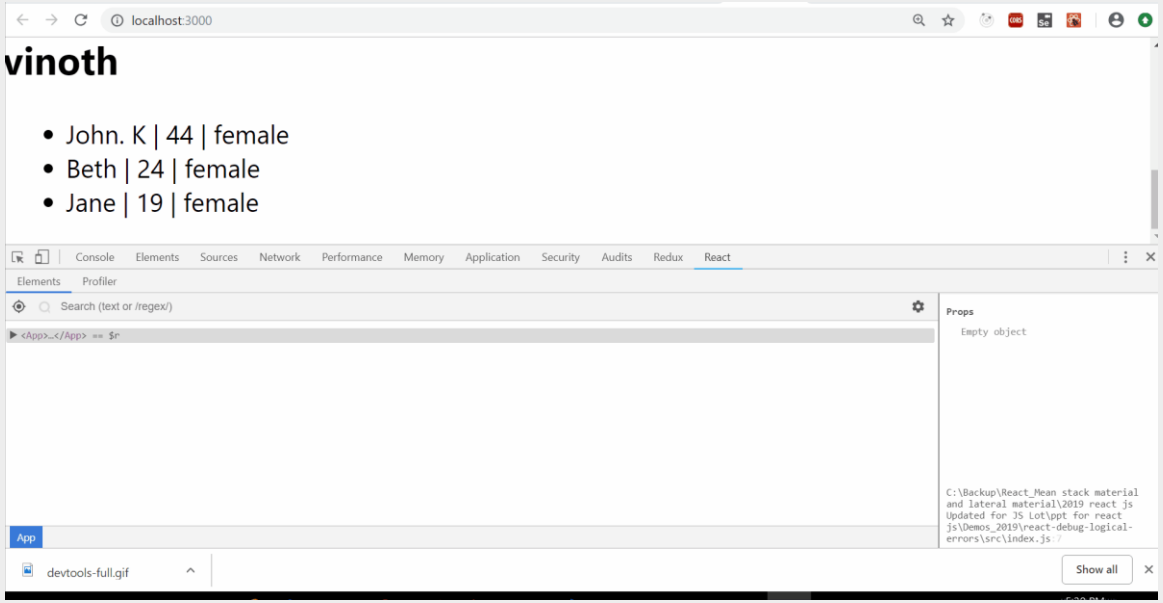
- Arrow keys or hjkl for navigation
- Right click a component to show in elements pane, scroll into view, show source, etc.
- Differently-colored collapser means the component has state/context.

In the given below snapshot, welcome component is selected. Here we are using react tabe in developer tools tab.Like this we can slect any component From the screen. Ensure that react is selected after inspecting the element you want.



Debugging React apps using google developer tools and React DevTool contd.

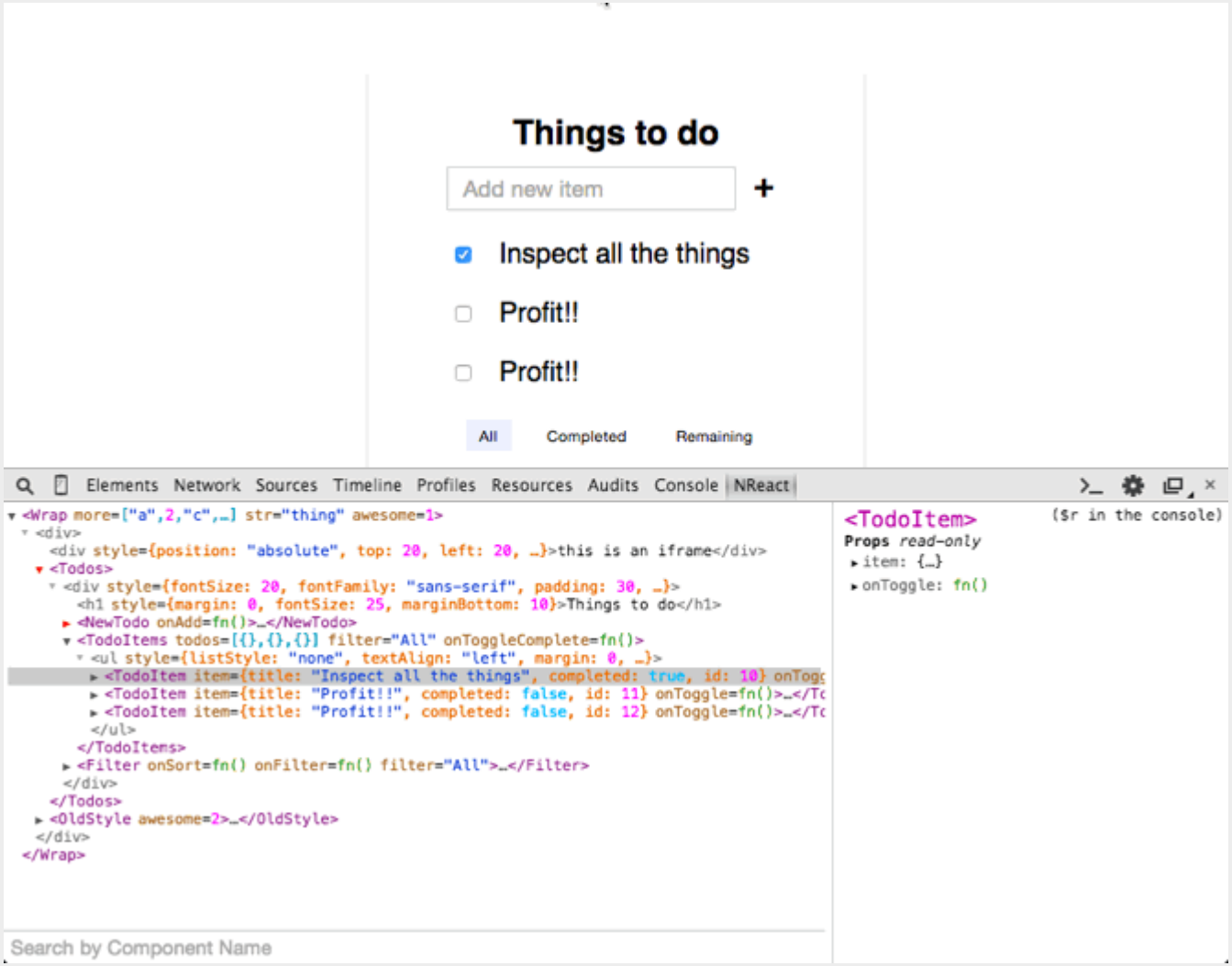
- Below is a sample demo of using react developer tools, press f5 to see animation.



Search Bar

Use the search bar to find components by name in react developer tools

Sample demo for using React Developer Tool



Demo

React-debug-logical-errors

Understanding Error Boundaries

- In Javascript we handle errors using try and catch as shown below and we can wrap a process that might return an error or exception in a try block and then offer a kind failsafe with the catch block after it has executed.

```
function doSomething (){
  try {
   return theExactThing();
   } catch (error) {
    return 'failsafe';
   }
 }
```

- **Error Handling in React**
  - With React we would expect to handle the error with similar syntax, something like this:

```
class ErrorBoundary extends React.component {
  render (){
   try {
     return <ExpectedOutput />
    } catch (error) {
     return <ErrorHandlerComponent />
     }
   }
 }
```

Understanding Error Boundaries contd.

So before React 16 came, we must have experienced a confusing and cryptic error messages while using React like this error snippet below:

```
TypeError: Cannot read property 'getHostNode' of null ?
 at getHostNode(~/react/lib/ReactReconciler.js:64:0)
 at getHostNode(~/react/lib/ReactCompositeComponent.js:383:0) ?
 at getHostNode(~/react/lib/ReactReconciler.js:64:0)
```

To solve this problem for React users, React 16 introduces a new concept of an **"error boundary"**.

Error boundaries:

Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed or logging the exact error.

Error boundaries catch errors during :

1. rendering, 2. life cycle methods, 3. constructors

Understanding Error Boundaries contd.

An Error Boundary in React is defined as a Class Component that defines one or both of **static getDerivedStateFromError() or componentDidCatch()** lifecycle methods.

static getDerivedStateFromError() :

This is used to render a fallback UI after an error has been thrown

componentDidCatch() :

- This is used to log error information.
- This works in the same way that Javascript's try/catch works.
- This method is invoked with the an error and info parameters that contain more context on the thrown error.

Where To Use Them

A top level Error Boundary can prevent the React Application from crashing due to unexpected problems and displaying a more apt message to end users.

Demo

React-error-boundaries-demo

Summary

- Understanding React Error Messages
- Handling Logical Errors,
- Debugging React apps using google developer tools and React DevTool
- Understanding Error Boundaries

Review Questions

- Question 1:
- _____is where you try to map URLs to destinations that aren't physical pages such as the individual views in your single-page app.
  - A) React
  - B) Route
  - C) Routing
  - D) Navigate
- Question 2:
- Which Command creates a frontend build pipeline, so you can use it with any backend you want.
  - A) React-App
  - B) create-react-app
    C) create-app
    D) React-app