

Instructor Notes:

Add instructor notes
here.




Instructor Notes:

Add instructor notes here.

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



LEARNING FACEBOOK'S
React.js

Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 2

Instructor Notes:

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 see the state of application which we generally print using `Console.log`

It also shows most 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;
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

3

Using `console.log`:

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

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;
```

`console.log` uses the reference when we expand it with the arrow down, but the value in the collapsed version. Keep that in mind when trying to debug React application—it might look like the valid data is passed to a child component while in fact it is being mutated in parent!

Instructor Notes:

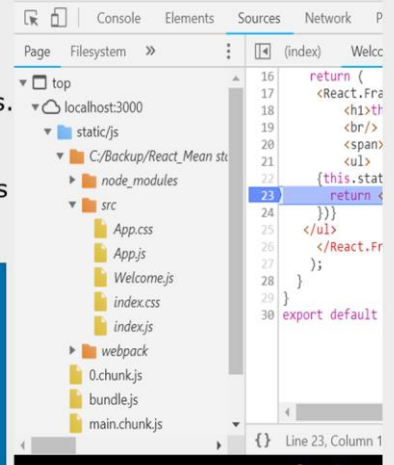
Handling Logical Errors

Logical errors are difficult to spot mostly in an application. Generally errors will be displayed in the browser screen and console, but logical errors won't.

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 named as "page" under which there is

```
localhost:3000 -> static/js ->
C:\Backup\
react js\Demos\react-debug-errors ->
src -> Welcome.js
(shown in screenshot on right)
```



Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

4

Errors are good things; they are feedback we get from malfunctions or bad inputs from users.

They shouldn't be ignored even when they don't have immediate side effects. Side effects may also occur in another render phase, and when it does it becomes increasingly hard to deal with.

Use the chrome developer tools.

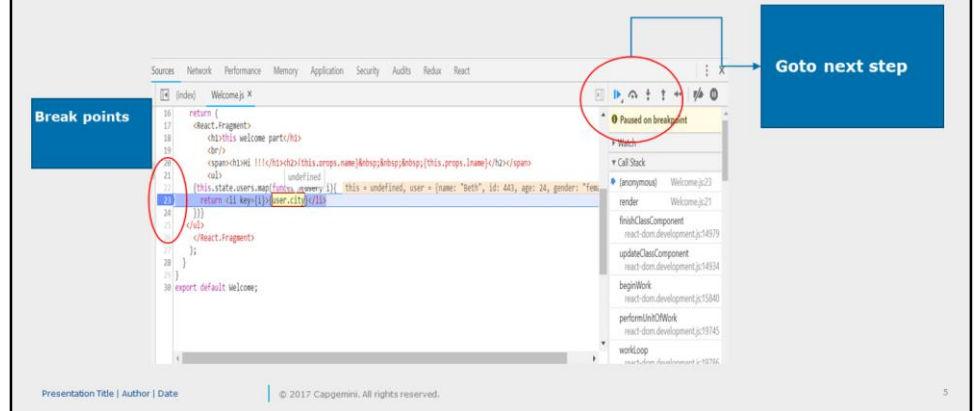
There you can go to the sources tab, in the sources tab you will find your code structure shown under localhost, there you will find a **src folder** and you actually find your code here.

Instructor Notes:

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.



Now this is possible due to source maps which are generated.

Basically you could say translation files allowing the browser developer tools to go into your code as you wrote it and allow you to debug that code even though the code which is shipped to the browser will be a different one, an optimized and bundled one.

This is pretty cool because you can debug the code you wrote even though it's not the code running in the browser.

So that's a cool tool you should be aware of for finding and fixing errors.

Using the developer tools with the sources tab where you can access your original code due to source maps.

Instructor Notes:

Add instructor notes here.

Demo



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



Add the notes here.

Instructor Notes:

Slide explains regarding the SOAP format

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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

7

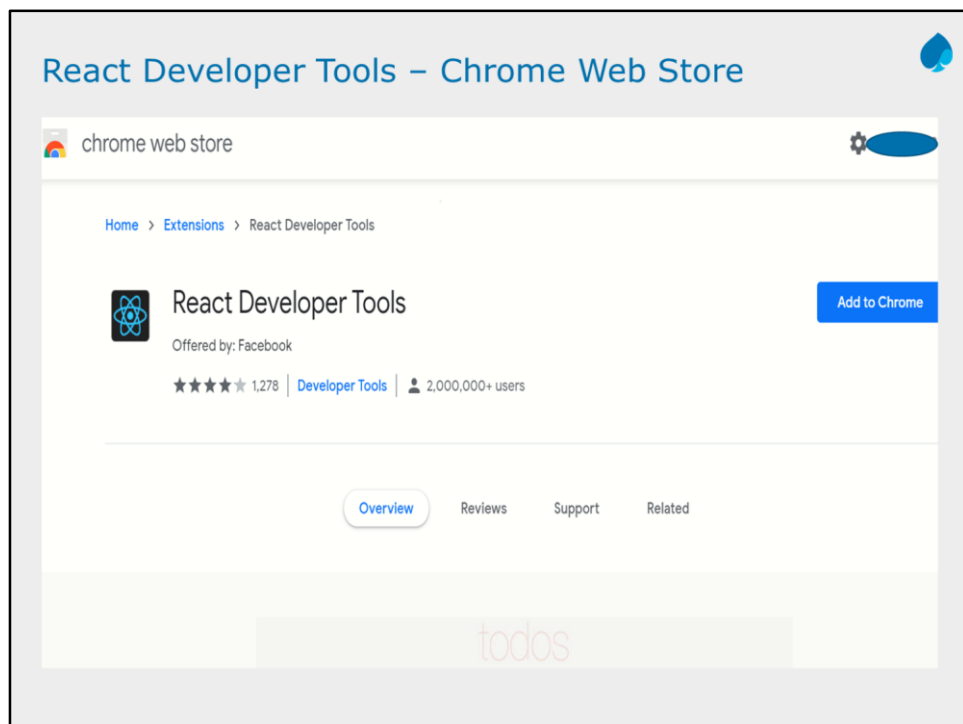
The react **Developer Tools** to install.

These are free in the Chrome Web Store.

You can click Add to chrome if you haven't done so.

Once you added them you probably need to restart chrome and thereafter in the Chrome developer tools.

You should find queue entries components and profiler with this react symbol next to them.

Instructor Notes:

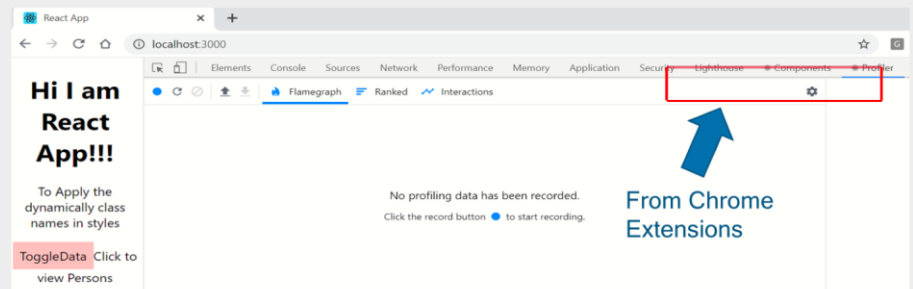
You will find this React Developer Tools to get installed in Chrome Web Store.

Instructor Notes:

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

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

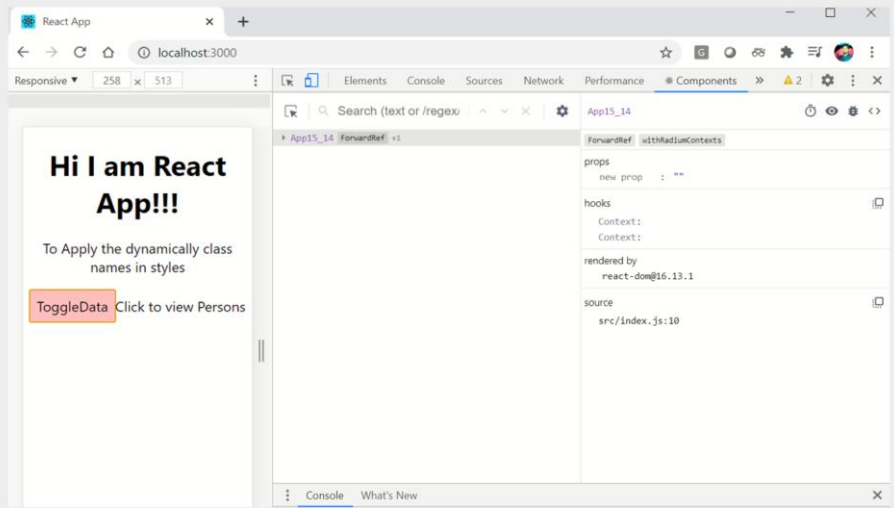
In the given below snapshot, welcome component is selected. Here we are using react tab in developer tools tab. Like this we can select any component From the screen. Ensure that react is selected after inspecting the element you want.



Instructor Notes:

Components Rendered – react Developer Tools

Currently rendered Components and its props and state will be displayed.

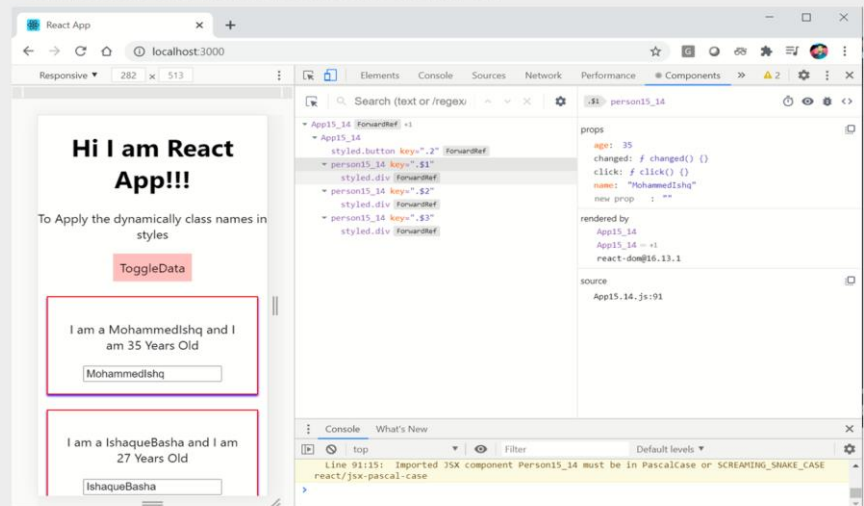


The screenshot displays the React Developer Tools interface. On the left, the application preview shows a web page with the text "Hi I am React App!!!" and a button labeled "ToggleData". The right panel shows the "Components" tab, which lists the currently rendered component tree. The selected component is "App15_14", which is a "ForwardRef" component. The props and hooks for this component are displayed on the right side of the panel. The props section shows "new prop : "" and the hooks section shows "Context:". The source code location is listed as "src/index.js:10".

Instructor Notes:

Components Rendered – react Developer Tools

You can now see the Currently rendered Person component from App Component and its properties are displayed.

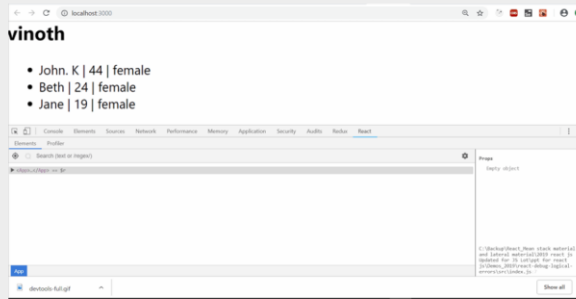


Instructor Notes:

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

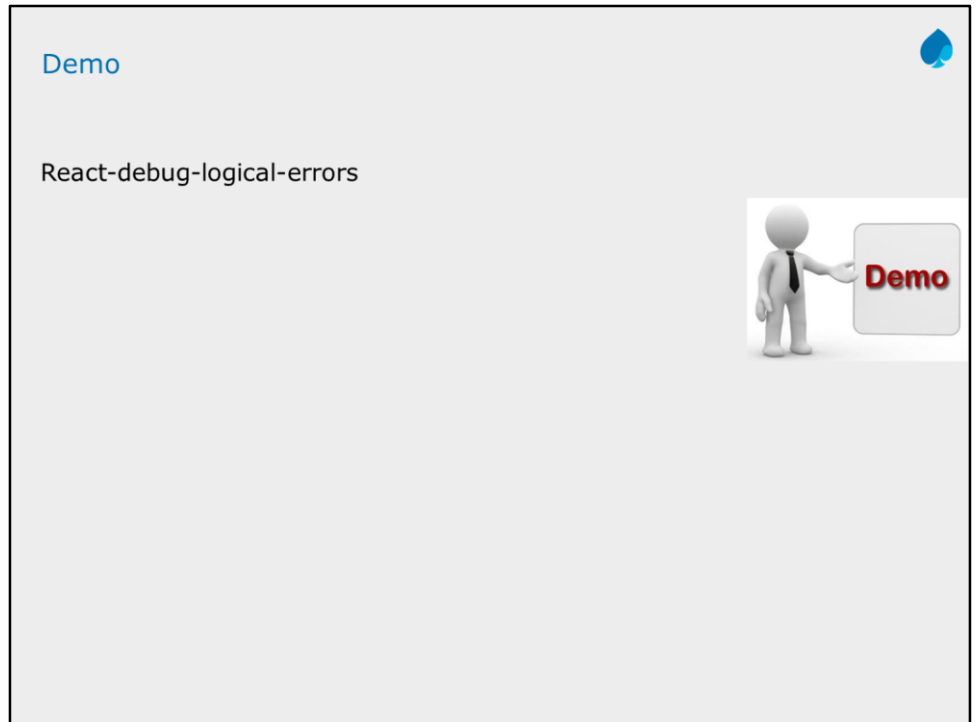
Use the search bar to find components by name in react developer tools

Instructor Notes:

[illegible]

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

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 />  
    }  
  }  
}
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

15

When you built a React app, you probably have experienced an error at some point that was cryptic and did not provide any meaningful context on what happened.

The error occurred at some point in the application and our React component did not handle the error gracefully, well mostly because it was none of its business to make sure the entire app holds.

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

An Error Boundary is a React component that catches errors within its children and does something meaningful with them such as post them to an error logging service or display a fallback UI for the specific child while maintaining the rest of the React app's sanity.

Therefore, for a block of functionality to be covered by Error Boundaries, it has to be a child of one in the first place.

Instructor Notes:

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

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

16

JavaScript errors inside components used to corrupt React's internal state and cause it to emit cryptic errors on next renders.

This could be due to the caching strategy of having to push from virtual to actual DOM so, the entire application crashes down

Error Boundaries:

Error boundaries do **not** catch errors for:

Event handlers

Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)

Serverside rendering

Errors thrown in the error boundary itself (rather than its children)

Instructor Notes:

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 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.

Instructor Notes:**ErrorBoundary**

➤Below we have created the exception to check how to handle the exception in React using error boundary.

```
const person = (props) => {  
  const errorDemo = Math.random();  
  if(errorDemo > 0.7)  
  {  
    return new Error('Something Went  
Wrong!!! ');  
  }  
  
  return (  
    //some code goes here...  
  )  
}
```

Instructor Notes:

Wrap Error Boundary with your Component



```
return (  
  <ErrorBoundary key={person.id}>  
    <Person15_15  
      click={ () =>  
        this.deletePersonHandler(index)  
      }  
      name={person.name}  
      age={person.age}  
      key={person.id}  
    />  
  </ErrorBoundary>  
)
```

➤ Wrap Error Boundary with your Component

➤ ErrorBoundary component is at now higher-level component.

➤ Whereas, Person component is as child component inside ErrorBoundary.

➤ So, errors of Person component will be handled by ErrorBoundary Component.

Instructor Notes:

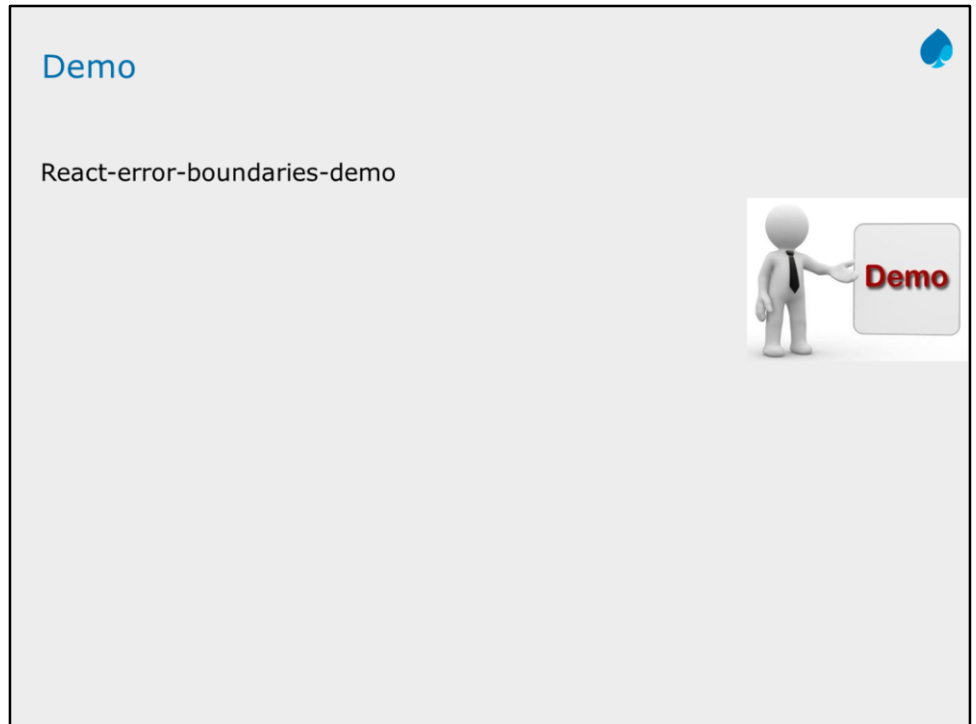
Error Boundary Key Value



- Error Boundary is so called Higher Order Component.
- As it is now Outer component, we need to move the key id to outer component Error Boundary
- Whereas, key always has to be on the outer element in a map method.

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

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



Add the notes here.

Instructor Notes:**Review Questions**

▪ Question 1:

▪ What is true w.r.t componentDidCatch()?

- A) used to log error information in error boundary
- B) used for debugging application
- C) it is not same as JS try and catch
- D) method cannot be called by passing error and info parameters

▪ Question 2:

_____ 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.

- A) Error Boundaries
- B) Exception Boundaries
- C) Error Handler
- D) JS Try-Catch