Lab 4
React Basics

CSCI2720 Building Web Applications

Agenda

- Tools to prepare
- Hello World!
- Component by component
- Looping through an array
- Events and states
- Conditional rendering

Tools to prepare

- You need to get these ready
  - Google Chrome
  - React Developer Tools
    - https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi
  - Web Server for Chrome
    - Otherwise you'll run into CORS issues

Hello World!

- Start with the given zip file here

  http://www.cse.cuhk.edu.hk/~chuckjee/2720lab4/lab4.zip

  - `index.html`
    - React, ReactDOM, Babel and Bootstrap are already included
  - `images`
    - Some nice pictures of CUHK to showcase in your React app later
- You need to prepare a JSX file app.jsx in the same directory
  - The names of `app.jsx` and `#app` in the HTML are arbitrary
  - You could use any name you wish in your own development

Hello World!

- In app.jsx, you only need one statement:

```
ReactDOM.render(
  <h1>Hello World</h1>,
  document.querySelector("#app")
);
```

  - This is the "entry point" of the React app
- Save the file and view the HTML via Web Server for Chrome (http://localhost:8887)
- Check out the Developer Tools with the new React tabs
  - There are no components yet

The first component

- Now, put this into your jsx file

```
class App extends React.Component
{
  render() {
    return <h1>Hello World</h1>;
  }
}
```

- And adjust your ReactDOM line to

```
ReactDOM.render(<App />,
document.querySelector("#app"));
```

  - This line must come after the definition of the App class, otherwise <App/> cannot be found

- This time, you should be able to see the component App with no props nor states

Now let's start with the app

- Our goal looks like this:

CUHK Pictures

cuhk-2013.jpg

Year: 2013

cuhk-2017.jpg

Year: 2017

sci-2013.jpg

Year: 2013

shb-2013.jpg

Year: 2013

stream-2009.jpg

Year: 2009

- You need to divide your app into components and build them one by one

Now let's start with the app

The <App/> class

- Use this code for your class App

```
class App extends React.Component {
  render() {
    return (
      <> /*** This fragment syntax is...*/
        <Header name={this.props.name}/>
        <Main />
        /* Header and Main are the React
         classes/components to build */
      </> /*** for more than 1 elements*/
    );
  }
}
```

- The `name` props comes from an "attribute" setting in the parent:

```
ReactDOM.render(
<App name="CUHK Pictures"/>,
document.querySelector("#app"));
```

  - You only need ONE line of ReactDOM.render()

The <Header/> class

- The Header inherits the `name` props from App (passing by parent)
- Here the styling is done by Bootstrap classes
  - Note: use `className` instead of `class` for the CSS classes!

```
class Header extends React.Component {
  render() {
    return (
      <header className="bg-warning">
        <h1 className="display-4 text-
center">{this.props.name}</h1>
      </header>
    );
    /* <header> here is an HTML tag! */
  }
}
```

The <Main/> class

- The Main class is merely a container for the contents we build later
- We better put some debugging text here before moving on

```
class Main extends React.Component {
  render() {
    return (
      <main className="container">
        Is this okay?
      </main> /* Main vs main? */
    );
  }
}
```

- You should be able to see the skeleton rendered, and more components are seen in the Developer Tools

Preparing the data

- Set up a simple data variable for the file information
  - *Hardcoding isn't a good idea for actual production!*

```
const data = [
  {filename: "cuhk-2013.jpg", year: 2013, remarks: "Sunset over CUHK"},
  {filename: "cuhk-2017.jpg", year: 2017, remarks: "Bird's-eye view of CUHK"},
  {filename: "sci-2013.jpg", year: 2013, remarks: "The CUHK Emblem"},
  {filename: "shb-2013.jpg", year: 2013, remarks: "The Engineering Buildings"},
  {filename: "stream-2009.jpg", year: 2009, remarks: "Nature hidden in the campus"},
];
```

  - An array of objects
  - This global **`const`** variable should be in the top of the file

Bootstrap cards

- We would like to show each image as a Bootstrap card
  - *Ref:* *https://getbootstrap.com/docs/5.0/components/card/#images-1*
  - For one card, the structure would be

```
<div className="card" …>
  <img src="..." …>
  <div className="card-body">
    <h6 className="card-title"…
    <p className="card-text">…
  </div>
</div>
```

- Can you set up `<FileCard/>` to show only `data[0]` first?
  - You should render `<FileCard/>` in `<Main/>`
  - Use `style={{width:200}}` for the card

Looping through the data array

- We want to show all images, so we will loop through the array
  - `.map()` is an efficient way to generate the result
    - `array.map( (value,key) => (every output) );`
- Adjust your return code in `<FileCard/> render()` to be

```
<>
{ data.map((file,index) => (
  <div key={index} className="card
d-inline-block m-2" style={{width:200}}>
    <img src={"images/"+file.filename}
alt=" {file.remarks}" className="w-100" />
    …
    /* file represents the iterator */
    …
</div> ))} /* mind the syntax and brackets! */
</> /* needed again due to multiple <div> */
```

- You need to be extremely careful with the syntax
- `key={index}` allows React to identify the elements in the ReactDOM for efficient re-rendering

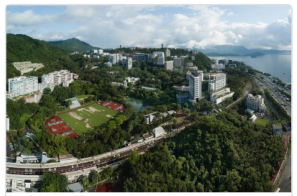
Careful output

- You should result at such a nice showcase of CUHK pictures

CUHK Pictures

cuhk-2013.jpg

Year: 2013



cuhk-2017.jpg

Year: 2017

sci-2013.jpg

Year: 2013

shb-2013.jpg

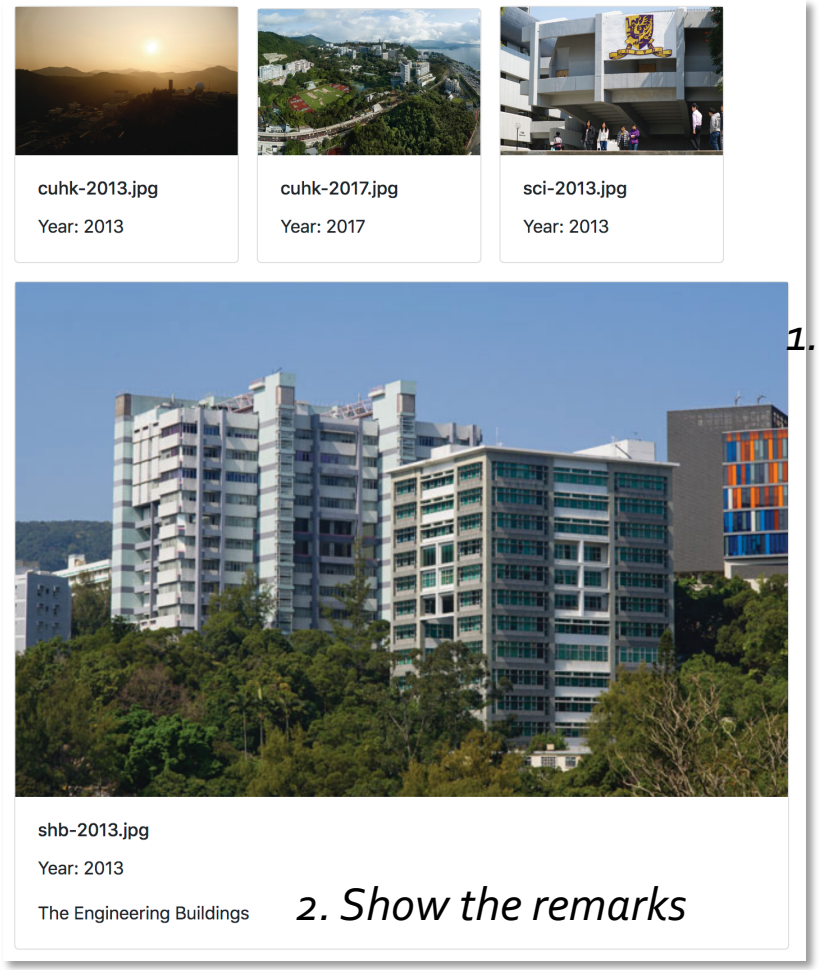
Year: 2013

stream-2009.jpg

Year: 2009

Handling events

- Let's do these when the user clicks on an image…



Handling events

- Set up an event handler *inside* <FileCard/>

```
class FileCare ... {
  handleClick() {
    console.log("clicked");
  }
  render () …
```

- And put the `onClick` handler in the card div

```
onClick={this.handleClick}
```

  - *The name handleClick isn't important as long as they match*

- Are you able to see a message when clicking?
- However, since we want to send the index too, you need to use this for `onClick`

```
onClick={(e) => this.handleClick(index,e)}
```

- And adjust the event handler

```
handleClick(index, e) {
  console.log(index);
}
```

- Are you able to see the index printed when clicking?

Using states

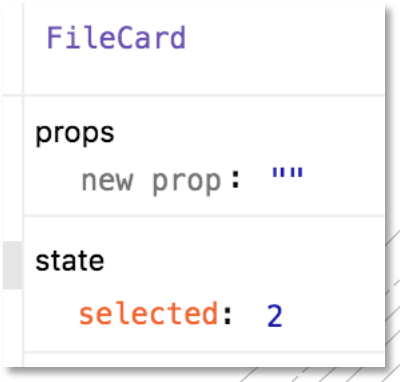
- To use states, you need to set it up in the class constructor of <FileCard/>

```
constructor(props) {
    super(props);
    this.state = { selected: -1 };
    /* this syntax should only be used
     in the constructor, and otherwise
     this.setState() must be used */
}
```

- In the event handler, you could do this (with proper JavaScript!) with `this.setState()`

```
If this.state.selected is not index
  set selected state to index
Else
  set selected state to -1
```

- Now, when clicking the cards you can see a change in the developer tools



Conditional rendering

- There are different ways to render conditionally in React
- Using ternary operator **?:**

```
style={{width:
this.state.selected==index ?
'100%' : 200}}
```

- Using logical operator **&&**

```
{ this.state.select==index &&
<p className="card-
text">{file.remarks}</p> }
```

- And sometimes you can use traditional if-else statements too if not inside a JSX statement

You've done it!

- This is a very simple React app you have built
- Observe carefully how ***responsive*** and ***interactive*** it can be
- Use React developer tools to check how the props and states are passed or changed

Submission

- No submission is needed for labs
- What you have done could be useful for your further exploration or the upcoming assignment
- **Please keep your own file safely**