GUls
10 / 09 16 / 19
Rahul Arya

Graphical User Interfaces

Various Platforms / Languages / Tools
- Android / iOS / Desktop / Web / ...  
- Java / Swift / C# / JavaScript / ...  
- Android Studio / Xcode / Visual Studio / WebStorm / ...

- What’s the common element?
- Component-level abstraction

Anatomy of a cat

Anatomy of a cat CATS

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Leaderboard</th>
<th>Buttons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Prompt</td>
<td>Leaderboard</td>
<td>Restart Button</td>
</tr>
<tr>
<td>Typing Input</td>
<td>Options</td>
<td>Headers</td>
</tr>
</tbody>
</table>

CS61A Autocorrected Typing Software

<table>
<thead>
<tr>
<th>WPM indicator</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.1</td>
<td>100.0</td>
<td>5:56</td>
</tr>
</tbody>
</table>

WPM Indicator
Accuracy Indicator
Time Indicator
Web Development

- Why web development? Easy to pick up, play around in your browser, runs on pretty much every device!
- HTML
  - Describes the organization of a web page
  - Made up of "tags" in a tree structure:
    ```html
    <body>
      <div attribute="value">
        Some text
        <div>
          Some more content
          <button>Click me!</button>
        </div>
      </div>
      <input />
    </body>
    ```
- JavaScript
  - At a high-level, similar-“ish” to Python
  - Just new syntax - semicolons, braces, indentation optional!
    ```javascript
    let x = 5;
    ```
    ```python
    x = 5
    ```
- CSS
  - Describes "style" / appearance of a website
  - Colors, animations, layout
  - Will not discuss further, since it’s specific to the web
    ```css
    [extra] If you’re interested, a great CSS tutorial is at MDN: https://developer.mozilla.org/en-US/docs/Web/CSS
    ```
What problems does React solve?

- Manipulating the DOM tree directly is a pain as it gets more complex.
- The "component tree" of our GUI doesn't line up with the DOM tree in the browser.

Solutions

- React **enforces abstraction barriers between components**
  - Each node in the "component tree" is its own class, so components can't depend on implementation details of other components.
- **Below the abstraction barrier**, React (efficiently) generates and updates the DOM tree as the component tree changes.

---

**React Components and JSX**

- React components must:
  - Inherit from React.Component
  - Have a `render()` method that describes its children / subtree
  - `render()` typically describes its subtree using JSX

Example:

```javascript
class WebPage extends React.Component {
  render() {
    return (
      <div>
        <Header />
        <Body />
      </div>
    );
  }
}
```

- **More JSX**

Render a list of components:

```javascript
class WebPage extends React.Component {
  render() {
    let bodyList = [];
    let i = 0;
    while (i < 3) {
      bodyList.push(<Body />);
      i += 1;
    }
    return (
      <div>
        <Header />
        {bodyList}
      </div>
    );
  }
}
```

Include an expression in JSX:

```javascript
class WebPage extends React.Component {
  render() {
    return (
      <div>
        <div>
          1 + 2 is {1 + 2}
        </div>
      </div>
    );
  }
}
```
Passing information to child components

- The parent component may need to pass information to the child components
- Solution: props
  - Props are essentially “arguments” for a component
  - Received by the component’s constructor
  - Stored in a dictionary in the attribute `this.props`

```jsx
class WebPage extends React.Component {
    render() {
        return (
            <div>
                <Header />
                <Button text="some text" />
            </div>
        );
    }
}
```

Passing information to child components

```jsx
class WebPage extends React.Component {
    let buttonList = [];
    let i = 0;
    while (i < 3) {
        buttonList.push(
            <button key="Button # + i">
                Button # {i}
            </button>
        );
        i += 1;
    }
    return (
        <div>
            <Header />
            {buttonList}
        </div>
    );
}
```

Responding to user input

- So far, we can display information, but not respond to interaction!
- Want code to run when the user does something e.g. clicks a button, types some text, etc.
- Solution: event handlers
  - Functions that are called when an “event” occurs - often some form of user interaction
  - Can be specified using JSX:
    ```jsx
    <button onClick={handleClick}>
        {this.props.text}
    </button>
    ```
  - `handleClick` will be called when the `<button>` is clicked

```jsx
class Button extends React.Component {
    let handleClick = () => {
        alert("Clicked! I am " + this.props.text);
    }
    render() {
        return (
            <div>
                <button onClick={handleClick}>
                    {this.props.text}
                </button>
            </div>
        );
    }
}
```

Persistent State

- We know how to call a function when an event happens
- But our functions don’t do anything persistent!
- We need to give our components some sort of memory

- In Python, we’d use an instance attribute
  - Initialized in the constructor
  - Updated in the event handler

- Problem
  - The component does not `re-render` - React does not know when to update an attribute
  - Can use the `forceUpdate()` method to fix

```jsx
class Button extends React.Component {
    state = {
        text: "Button Text"
    }
    handleClick = () => {
        this.setState({text: "Updated Text"});
    }
    render() {
        return (
            <div>
                <button onClick={this.handleClick}>
                    {this.state.text}
                </button>
            </div>
        );
    }
}
```
Responding to user input

```javascript
class Button extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      numberOfClicks: 0
    };
  }

  handleClick() {
    this.setState({
      numberOfClicks: this.state.numberOfClicks + 1
    });
  }

  render() {
    return (<button onClick={this.handleClick}>Clicked {this.state.numberOfClicks} times !</button>);
  }
}
```

[Demo]

Persisting State

- **State** is stored in the `this.state` instance attribute, initialized in the constructor.
- Updated using the `this.setState()` method, so React knows when updates happen.

```
let handleClick = () => {
  this.setState({
    numberOfClicks: this.state.numberOfClicks + 1
  });
};
```

Event Handlers as Props

- Often, we want the parent component to update its state in response to an event handler on the child.
- Example: When a button is clicked, the header should update a counter.
- Event handler must be in the parent component to update state.
- But must be bound to an element in the child component.

  **Solution:** Pass the event handler as a prop to the child.

```
let handleClick = () => {
  this.props.onClick();
};
```

Responding to user input

```javascript
class WebPage extends React.Component {
  ...;
}
```

[Demo]
Summary + Thinking in React

- Directly manipulating the DOM tree gets complicated and messy fast - better to deal with a GUI as a tree of isolated components
- Components are classes that inherit from React.Component and that have a render() method
- Abstraction barriers isolate implementation of each component
- React updates the DOM tree below the abstraction barrier

- Data flows down the component tree in the form of props
- User input is captured using event handlers
- State is updated using setState() so React knows to re-render the DOM Tree
- Event handlers can be passed down the tree as props for events to flow up the component tree

Next Steps

- Interested in React / GUIs? Awesome!
- Check out the cats project GUI at https://github.com/Cal-CS-61A-Staff/cats-gui

- MDN JavaScript tutorial is a good, rigorous introduction to JavaScript for a 61A student
  - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps
- Official React tutorial is excellent, goes into a lot more depth
  - https://reactjs.org/
- Resources are available for Android / iOS development as well