Tutorial 11: ReactJS (2)

Objectives

In this tutorial, we will continue with the flashcards-react app & focus on practicing:

- Architecting your app
  - Decomposing App into components
  - Making decision(s) on where state(s) should live
- Using props
- Fetching data
- Using ***if*** in JSX

Discussions

Discussion 1: How to decompose flashcards app into components? (10mins)

React targets to create reusable & relatively small components. What are the components our flashcards app may have?

Discussion 2: Where does state live? (15mins)

In the flashcards app, we capture (1) a list of cards, (2) the index of the displaying card. They are all now in the top component App.

- How does this change after introducing new component(s) from *Discussion 1*?
- What data to pass into inner component(s)?

Note: **Lifting state up**

- *To collect data from multiple children, or*
- *To have two child components communicate with each other,*

→ *you need to declare the shared state in their parent component instead. The parent component can pass the state back down to the children by using props; this keeps the child components in sync with each other and with the parent component.*

Tutorial Exercises

Download the *tut10_solution: flashcards-react* & complete the exercises below.

Exercise 1: new components (20 mins)

- Implement the new introduced component(s) in Discussion 1 & 2.

Hint:

- Passing event handler into child component? – yes! props, just like other data

Exercise 2: events: flip card (15 mins)

- Using CSS class hidden to toggle show/ hide between word & definition (flip)
- Which component should the event handler belong to?

Exercise 3: Dynamic data (30 mins)

Using the API: https://wpr-quiz-api.herokuapp.com/cards for a list of cards & display.

This requires usage of method ***componentDidMount()*** – more information later in the next lecture.

```
async componentDidMount() {
    const response = await fetch('https://jsonplaceholder.typicode.com/users');
    const monsters = await response.json();

    this.setState({
        monsters: monsters
    });
}
```

IFY

IFY 1: Developer Tools

The React Devtools extension for Chrome and Firefox lets you inspect a React component tree with your browser's developer tools.

- The React DevTools let you check the props and the state of your React components.
- After installing React DevTools, you can right-click on any element on the page, click "Inspect" to open the developer tools, and the React tabs ("⚛ Components" and "⚛ Profiler") will appear as the last tabs to the right. Use "⚛ Components" to inspect the component tree.