TW-008 TEAM LEAD VERSION (Sprint-5 Week-2)

TEAMWORK

CLARUSWAY
WAY TO REINVENT YOURSELF

Meeting Agenda

- ▶ Icebreaking
- ▶ Questions
- ▶ Interview Questions
- ▶ Coding Challenge
- ▶ Video of the week
- ▶ Retro meeting
- ▶ Case study / project

Teamwork Schedule

Ice-breaking 5m

- Personal Questions (Study Environment, Kids etc.)
- Any challenges (Classes, Coding, studying, etc.)
- Ask how they're studying, give personal advice.
- Remind that practice makes perfect.

Team work 5m

- Ask what exactly each student does for the team, if they know each other, if they care for each other, if they follow and talk with each other etc.

Ask Questions 15m

1. When might you use `React.PureComponent`?

A. when you do not want your component to have props
B. when you have sibling components that need to be compared
C. when you want a default implementation of `shouldComponentUpdate()
D. when you do not want your component to have state

Answer: C

2. You have written the following code but nothing is rendering. How do you fix this problem?

```
const Heading = () => {
  <h1>Hello!</h1>;
};
```

A. Add a render function.
B. Change the curly braces to parentheses or add a return statement before the `h1` tag.
C. Move the `h1` to another component.
D. Surround the `h1` in a `div`.

Answer: B

3. Which option is correct for State vs Props

A. Props is something that the parent doesn't need and decide to throw around among other parents; State is the parent's favorite child and something the component wants to nurture.
B. Props is a copy of real DOM; State is the definition of the real DOM.
C. Props get passed to the component using naming conventions, like a function parameter; State is managed within the component and holds some information that may change over the lifetime of the component.
D. Prop is managed within the component and holds some information that may change over the lifetime of the component; State gets passed to the component, like a function parameter

Answer: C

4. What package contains the render() function that renders a React element tree to the DOM?

A. `React`
B. `ReactDOM`
C. `Render`
D. `DOM`

Answer: B

5. Which of the following click event handlers will allow you to pass the name of the person to be hugged?

```
class Huggable extends React.Component {
  hug(id) {
    console.log("hugging " + id);
  }

  render() {
    let name = "kitteh";
    let button = // Missing Code
    return button;
  }
}
```

A. `<button onClick={(name) => this.hug(name)}>Hug Button</button>`
B. `<button onClick={this.hug(e, name)}>Hug Button</button>`
C. `<button onClick={(e) => hug(e, name)}>Hug Button</button>`
D. `<button onClick={(e) => this.hug(name,e)}>Hug Button</button>`

Answer: D

6. Which answer best describes a function component?

A. A function component is required to create a React component.
B. A function component accepts a single props object and returns a React element.
C. A function component is the only way to create a component.
D. A function component is the same as a class component.

Answer: D

7. What is the name of the tool used to take JSX and turn it into createElement calls?

A. JSX Editor
B. ReactDOM
C. Browser Buddy
D. Babel

Answer: D

8. If you see the following import in a file, what is being used for state management in the component?

```
import React, {useState} from 'react';
```

A. React Hooks
B. stateful components
C. math
D. class components

Answer: A

9. If you created a component called Dish and rendered it to the DOM, what type of element would be rendered?

```
function Dish() {
  return <h1> Mac and Cheese</h1>;
}

ReactDOM.render(
    <Dish />
    document.getElementById('root')
)
```

A. div
B. section
C. component
D. h1

Answer: D

10. Which of the following lifecycle methods does not get triggered on the component's initial render?

A. componentWillMount()
B. componentWillReceiveProps()
C. render()
D. componentDidMount()

Answer: B

11. What does this React element look like given the following function?

```
React.createElement(
    "h1",
    null,
    "What's happening?"
);
```

A.

```
<h1 props={null}>What's happening?</h1>
```

B.

```
<h1 id="component">What's happening?</h1>
```

C.

```
<h1>What's happening?</h1>
```

D.

```
<h1 id="element">What's happening?</h1>
```

Answer: C

Interview Questions 15m

1. What are Promises?

Answer: Promises are one way in handling asynchronous operations in JavaScript. It represents the value of an asynchronous operation. Promises was made to solve the problem of doing and dealing with async code before promises we're using callbacks.

Promises have 3 different states.

Pending - The initial state of a promise. The promise's outcome has not yet been known because the operation has not been completed yet.

Fulfilled - The async operation is completed and successful with the resulting value.

Rejected - The async operation has failed and has a reason on why it failed.

Settled - If the promise has been either Fulfilled or Rejected.

The Promise constructor has two parameters which are functions resolve and reject respectively. If the async operation has been completed without errors call the resolve function to resolve the promise or if an error occurred call the reject function and pass the error or reason to it.

2. What is a Callback function?

Answer: A Callback function is a function that is gonna get called at a later point in time.

```
const btnAdd = document.getElementById('btnAdd');

btnAdd.addEventListener('click', function clickCallback(e) {
    // do something useless
});
```

In this example, we wait for the click event in the element with an id of btnAdd, if it is clicked, the clickCallback function is executed. A Callback function adds some functionality to some data or event. The reduce, filter and map methods in Array expects a callback as a parameter. A good analogy for a callback is when you call someone and if they don't answer you leave a message and you expect them to callback. The act of calling someone or leaving a message is the event or data and the callback is the action that you expect to occur later.

3. What are controlled components?

Answer: In HTML, form elements such as "input", "textarea", and "select" typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. A form element whose value is controlled by React in this way is called a "controlled component". With a controlled component, every state mutation will have an associated handler function. This makes it straightforward to modify or validate user input.

4. What is a higher order component?

Answer: A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API. They are a pattern that emerges from React's compositional nature. A higher-order component is a function that takes a component and returns a new component. HOC's allow you to reuse code, logic and bootstrap abstraction. HOCs are common in third-party React libraries. The most common is probably Redux's connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

5. How Virtual-DOM is more efficient than Dirty checking?

Answer: In React, each of our components have a state. This state is like an observable. Essentially, React knows when to re-render the scene because it is able to observe when this data changes. Dirty checking is slower than observables because we must poll the data at a regular interval and check all of the values in the data structure recursively. By comparison, setting a value on the state will signal to a listener that some state has changed, so React can simply listen for change events on the state and queue up re-rendering. The virtual DOM is used for efficient re-rendering of the DOM. This isn't really related to dirty checking your data. We could re-render using a virtual DOM with or without dirty checking. In fact, the diff algorithm is a dirty checker itself. We aim to re-render the virtual tree only when the state changes. So using an observable to check if the state has changed is an efficient way to prevent unnecessary re-renders, which would cause lots of unnecessary tree diffs. If nothing has changed, we do nothing.

Coding Challenge 20m

- Coding Challenge: Bracket Validator (JS-06)

☕

Coffee Break 10m

☕

Video of the Week 5m

- How to prepare for a job interview

Retro Meeting on a personal and team level 5m

Ask the questions below:

- What went well?
- What went wrong?
- What is the improvement areas?

Case study/Project 15m

Case study should be explained to the students during the weekly meeting and has to be completed in one week by the students. Students should work in small teams to complete the case study.

- Tour Places (RP-01)

Closing 5m

-Next week's plan

-QA Session
