goto

goto impact foundation

Generasi Gigih

Full Stack Engineer

Module 4 Session 3: Rendering List and Event Handling

Our Agenda

Warm Up

Pre-Assignment Showcase

Core Material Lecture

Individual/ group/pair practice & debriefing:

QnA & Homework

Wrap Up

Generasi Gigih

Let’s Warm Up!

What did you guys learned on session 2?

Generasi Gigih

Showcase Time!

Anyone wants to volunteer?

Let’s Discuss

1. Rendering list
2. Event handling
3. Event propagation

Generasi Gigih

Rendering List

Array of data to array of components

Part 1: JS map function

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}
```

Part 2: JS filter function

```
const ages = [32, 33, 16, 40];
const result = ages.filter(checkAdult);

function checkAdult(age) {
  return age >= 18;
}
```

List

```
<ul>
  <li>Mangga: Kuning</li>
  <li>Apel: Merah</li>
  <li>Alpukat: Hijau</li>
  <li>Anggur: Ungu</li>
  <li>Pisang: Kuning</li>
</ul>
```

- The only difference among those list items is **their contents, their data.**
- you can **store that data in JavaScript objects and arrays** and
- Then use methods like **map() and filter() to render lists** of components from them.

Part 1: Storing data

```
const fruits = [
  'Mangga: Kuning',
  'Apel: Merah',
  'Alpukat: Hijau',
  'Anggur: Ungu',
  'Pisang: Kuning'
];
```

We store the content/ data into an array

Part 2: Map the array into a new array of JSX nodes

```
const listItems = fruits.map(fruit => <li>{fruit}</li>);
```

- Map the array
- Return each item wrapped in jsx in the array
- Store it into a new variable listItems

Any question?

Part 3: Return listItems from your component wrapped in a

```
return <ul>{listItems}</ul>;
```

Render it within a <ul> tag

Part 4: Complete code

```
const fruits = [
  'Mangga: Kuning',
  'Apel: Merah',
  'Alpukat: Hijau',
  'Anggur: Ungu',
  'Pisang: Kuning'
];

export default function List() {
  const listItems = fruits.map(fruit =>
    <li>{fruit}</li>
  );
  return <ul>{listItems}</ul>;
}
```

Try to see and understand it in a whole

Hands on

Hands on 1: Render list of the members in your family

- Try to create a list of your family members (name and gender) using the previously learned approach
- You fill find an error message in the console, what is it?

```
Warning: Each child in a list should have a unique "key" prop.

Check the render method of `List`. See
https://reactjs.org/link/warning-keys for more information.
    at li
    at List
```

You will learn about this error later. Now we will learn on how to filter the data first before jumping in to this error

Hands on 2: Structure your data

```
const family = [{
  id: 0,
  name: 'Papa Shark',
  gender: 'male'
}, {
  id: 1,
  name: 'Mama Shark',
  gender: 'female'
}, {
  id: 2,
  name: 'Teenager Shark',
  gender: 'male'
}, {
  name: 'Baby Shark',
  gender: 'female'
}]
```

If you haven't, add structure to your data like this so each object has its **id, name, and gender**

Hands on 3: Filtering your data

- Try to show only female members of your family with filter function
- Clue:
  - Filter it to a new variable femaleMembers
  - Map this femaleMembers to a new variable (listItems for example)
  - Render the variable

- Did you successfully complete the task?
- Here is the complete code, try to go through and understand it

```
const family = [{
  id: 0,
  name: 'Papa Shark',
  gender: 'male'
}, {
  id: 1,
  name: 'Mama Shark',
  gender: 'female'
}, {
  id: 2,
  name: 'Teenager Shark',
  gender: 'male'
}, {
  name: 'Baby Shark',
  gender: 'female'
}]
```

```
export default function List() {
  const femaleMembers = family.filter(member =>
    member.gender === 'female'
  );
  const listItems = femaleMembers.map(member =>
    <li>
      {member.name}
    </li>
  );
  return <ul>{listItems}</ul>;
}
```

Notice that this code also has error

Hands on 4: Adding keys

Console

Warning: Each child in a list should have a unique “key” prop.

```
<li key={member.id}>...</li>
```

Try adding key on each

Part 5: Understanding keys (1)

- JSX elements directly inside a **map() call always need keys**
- Keys tell React which array item each component corresponds to (so it can match them up)
- This is **important** when your array items can move (e.g. due to sorting), get inserted, or get deleted
- Keys help React infer what exactly has happened, and **make the correct updates to the DOM tree**

Part 5: Understanding keys (2)

- Rules:
  - Keys **must be unique among siblings**. However, it's okay to use the same keys for JSX nodes in different arrays.
  - **Keys must not change** or that defeats their purpose

- Why need keys?
  - Imagine that files on your desktop didn't have names
  - You identify them by order (first, second, third)
  - Once you delete a file, it would get confusing. The second file would become the first file, third becomes second, and so on

Part 6: Notes for keys

- **Avoid** to use an **item's index in the array** as its key → this is what React use when you don't specify a key, as said before it will get confusing if using order

- **Do not generate keys on the fly** (e.g. with key={Math.random()}) → This will cause **keys to never match up between renders**, leading to all your components and DOM being **recreated every time**

- **Your components won't receive key as a prop** → if you need ID, pass it as a separate props

Part 7: Recap! Make sure you understand...

- How to move data out of components and into data structures like arrays and objects.
- How to generate sets of similar components with JavaScript's map().
- How to create arrays of filtered items with JavaScript's filter()
- Why and how to set key on each component in a collection so React can keep track of each of them even if their position or data changes

Hands on

Rendering list

Do not forget to use keys

https://codesandbox.io/s/handson-rendering-list-ckjy86

Generasi Gigih

Event Handling (hands on)

Responding to interactions

Part 1: Introduction

- React lets you add event handlers to your JSX
- Event handlers are **your own functions** that will be triggered in **response to interactions** like clicking, hovering, focusing form inputs, and so on

```
export default function Button() {
  return (
    <button>
      I don't do anything
    </button>
  );
}
```

This is a button that doesn’t do anything yet

Let’s add event handler to this button

We will go through the slide while exercising on codesandbox

Generasi Gigih

Hands on

https://codesandbox.io/s/hands-on-eventhandling-r3vn24

Hands on 1: Handler function

You can make it show a message when a user clicks by following these three steps:

a. Declare a function called handleClick inside your Button component.

b. Implement the logic inside that function (use alert to show the message)

c. Add `onClick={handleClick}` to the `<button>` JSX

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Part 2: What did you just do?

1. You defined the handleClick function and then **passed it as a prop to <button>**
2. handleClick is an event handler function

3. Event handler functions:
   a. Usually defined inside your components.
   b. Have names that start with handle, followed by the name of the event.
   c. By convention, it is common to name event handlers as handle followed by the event name.

   E.g: onClick={handleClick}, onMouseEnter={handleMouseEnter}, and so on.

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Part 3: Correct vs incorrect handlers (read first)

Correct	Incorrect
<button onClick={handleClick}>	<button onClick={handleClick()}>
<button onClick={() => alert('...')}>	<button onClick={alert('...')}>

What is the difference between correct and incorrect one? Anyone?

Correct: **passing** a function

Incorrect: **calling** a function

Part 4: Reading props in event handlers

```
function AlertButton({ message, children }) {
  return (
    <button onClick={() => alert(message)}>
      {children}
    </button>
  );
}

export default function Toolbar() {
  return (
    <div>
      <AlertButton message="Playing!">
        Play Movie
      </AlertButton>
      <AlertButton message="Uploading!">
        Upload Image
      </AlertButton>
    </div>
  );
}
```

1. Because event handlers are declared inside of a component, **they have access to the component's props**
2. Go through and try to understand the code shown on this slide
3. Try it on the same codesandbox (**App2.js**)

Hands on 2: Pass handler function as props

1. Starts in **App3.js**
2. You see 2 event handlers functions
   a. handleUpload
   b. handlePlay
3. Pass these functions as props on AlertButton:
   a. Play → handlePlay
   b. Upload → handleUpload
4. Output:
   a. Play button alerts playing
   b. Upload button alerts uploading

```
function AlertButton({ children, onClick }) {
  return <button onClick={onClick}>{children}</button>;
}

function handlePlay() {
  alert("Playing!");
}

function handleUpload() {
  alert("Uploading");
}

export default function Toolbar() {
  return (
    <div>
      <AlertButton onClick={handlePlay}>Play Movie</AlertButton>
      <AlertButton onClick={handleUpload}>Upload Image</AlertButton>
    </div>
  );
```

Recap:

- You can handle events by passing a function as a prop to an element like <button>.
- Event handlers must be passed, not called! onClick={handleClick}, not onClick={handleClick()}.
- You can define an event handler function separately or inline.
- Event handlers are defined inside a component, so they can access props.

Generasi Gigih

Event Propagation

Stopping event propagation and prevent default

Generasi Gigih

Open this sandbox

https://codesandbox.io/s/hands-on-eventbubbling-sr253h?file=/src/App.js

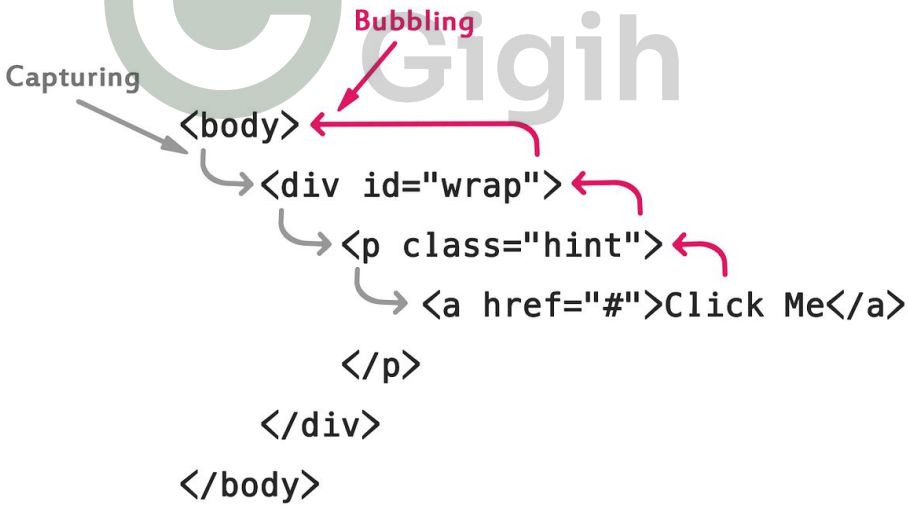
Part 1: Try on the codesandbox

- contains two buttons.
- Both the and each button have their own onClick handlers.
- **Which handlers do you think will fire when you click a button?**

- If you click on either button, its onClick will run first, **followed by the parent 's onClick** → 2 messages
- If you click **the div itself,** only the parent 's onClick will run → 1 message

Part 2: Event Propagation (1)

- Event propagation in Javascript (not only React JS)
  - Capturing phase – the event goes down to the element.
  - Target phase – the event reached the target element.
  - Bubbling phase – the event bubbles up from the element to its parent and so on



Part 2: Event Propagation (2)

- **Almost** all events bubble. For instance, a focus event does not bubble
- Further reading outside class:
  - Events that don't bubble: https://en.wikipedia.org/wiki/DOM_event#Events
  - More on event propagation: https://javascript.info/bubbling-and-capturing

Hands on: Stopping propagation

- Open App2.js
- Try to understand it and go through the code with mentor

- Try adding **e.stopPropagation();** in the event handler function on code sandbox
- e.stopPropagation() **prevents the event from bubbling further**

Hands on: Prevent Default

- Open App3.js
- Try to understand it and go through the code with mentor

- Try adding **e.preventDefault();** in the event handler function on code sandbox
- See the difference between with and without preventDefault

Hands on

Hands on 1: Props and Conditional Rendering

1. Create a component called clock that retrieve props **time and color**
2. Return **h1 with props time** and **style it with color from props**
3. Create a parent component that retrieve input color,
4. Call the Clock component, pass the inputted color and current date time as props
5. **Bonus point: add heart emoji if color is light coral**

Hands on 2: Handle Event and Stop Propagation

- **Link:** https://codesandbox.io/s/hands-on-events-exercise-gf7l36?file=/ColorSwitch.js

- Instruction:
  - Wire the button to the onChangeColor event handler prop it receives from the parent so that clicking the button changes the color.

  - Notice that clicking the button also increments the page click counter. Fix it so that clicking the button **only changes the color, and does not increment the counter**

Generasi Gigih

What we learned

1. Rendering List
2. Event handling
3. Event propagation

Generasi Gigih

Q&A!

Generasi Gigih

Homework

Try to polish your spotify application by breaking down components and use what we have learned so far!

goto

goto impact foundation

Generasi Gigih

See you on the next module!