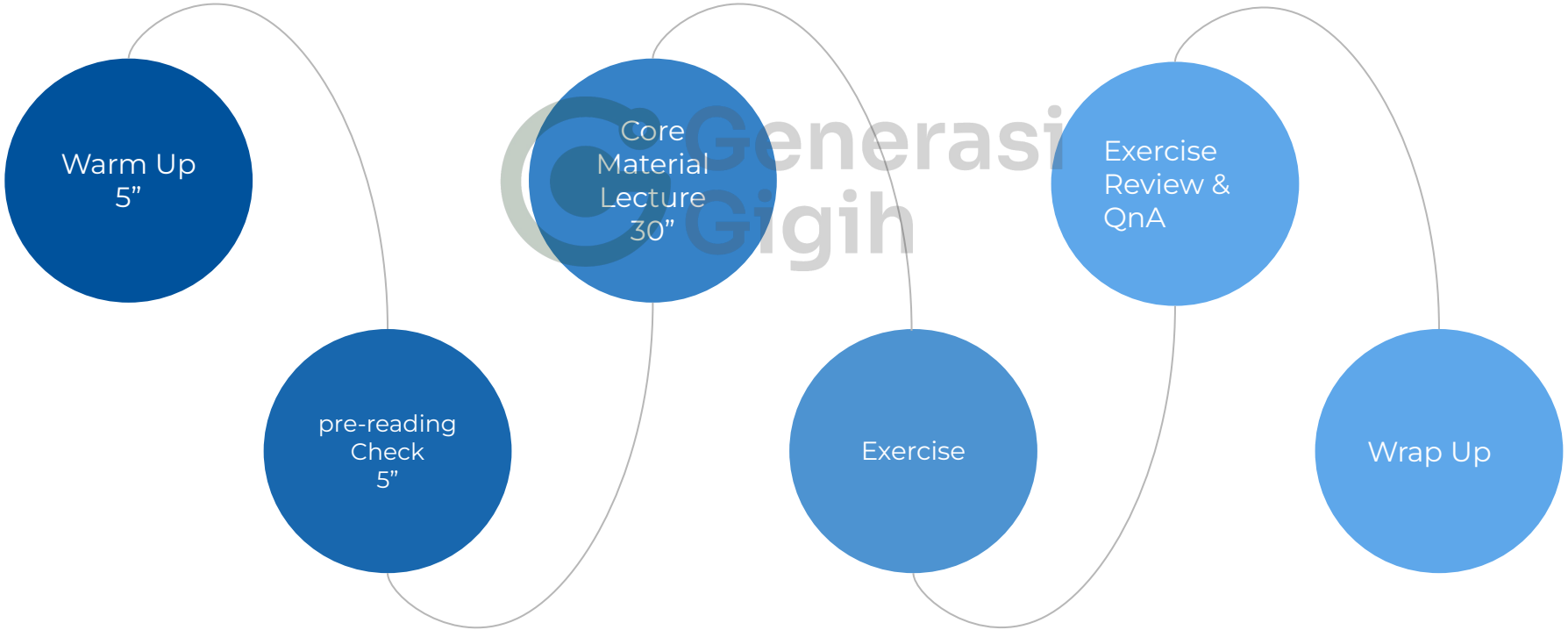
Full Stack Engineer

Module 5.1: Hooks

Our Agenda


Warm Up
5”
pre-reading
Check
5”
Core
Material
Lecture
30”
Exercise
Exercise
Review &
QnA
Wrap Up

Generasi Gigih

Let’s Warm Up!

Recall how does state work, and how to write the code

Let’s Discuss

- Hooks (10”)
  - Built in Hooks
- Lifecycle Functional Component (10”)
  - Difference lifecycle class and functional component
- ContextAPI (5”)
  - Props drilling solved!
- Custom Hooks (5”)
  - useFetch as example

Generasi Gigih

Let’s Talk About The Materials

Hooks

Hooks

- In previous versions of react (prior to v16.8), only Class component can use state feature
- Starting from v16.8, Functional component can use state using useState API from React
- Hooks offers more features, like encapsulating state logic inside a function
- Getting rid of Class usage because this keyword can be confusing

Built-in React Hooks

- useState (equivalent with this.state and this.setState)
- useEffect (equivalent to componentDidMount)
- useContext (as State Management)
- ... any many more

Hooks: useState

```jsx
import { useState } from 'react';

const Sample = () => {
  const [counter, setCounter] = useState(0);

  return (
    <>
      Counter is {counter}
      <br />
      <button onClick={() => setCounter(counter + 1)}>+</button>
      <button onClick={() => setCounter(counter - 1)}>-</button>
    </>
  );
};
```

useState API returns 2 objects:
- state
- state setter

It's up to us to name those objects, but the convention is `state` and `setState`

Equivalent to `this.state.counter`.
Access the state using the variable declared from `useState`

Equivalent to `this.setState({counter:})`.
Update the state using state setter variable declared from `useState`

Code

componentDidMount

We do network call to get users once component is rendered

```
class Sample extends Component {
  state = { users: [] };

  componentDidMount() {
    this.getUsers();
  }

  getUsers = async () => {
    const users = await fetch(
      'https://jsonplaceholder.typicode.com/users'
    ).then(response => response.json());
    this.setState({ users });
  };

  render() {
    const { users } = this.state;

    return (
      <>
        {users.length > 0 && (
          <>
            <div> User list:</div>
            <ul>
              {users.map((user, index) => (
                <li key={index}>
                  {user.name} ({user.email})
                </li>
              ))}
            </ul>
          </>
        )}
      </>
    );
  }
}
```

- React component class has a `componentDidMount()` method that is useful to run a side effect operation that can alter the state
- Side effect operation:
  - Modifying DOM (i.e. accessing `document.title`)
  - Network calls
- `componentDidMount()`, as the name suggests, will run once the component is mounted or in other words rendered
- This is useful when we want to automatically do an operation once the page is loaded

Code

Hooks: useEffect

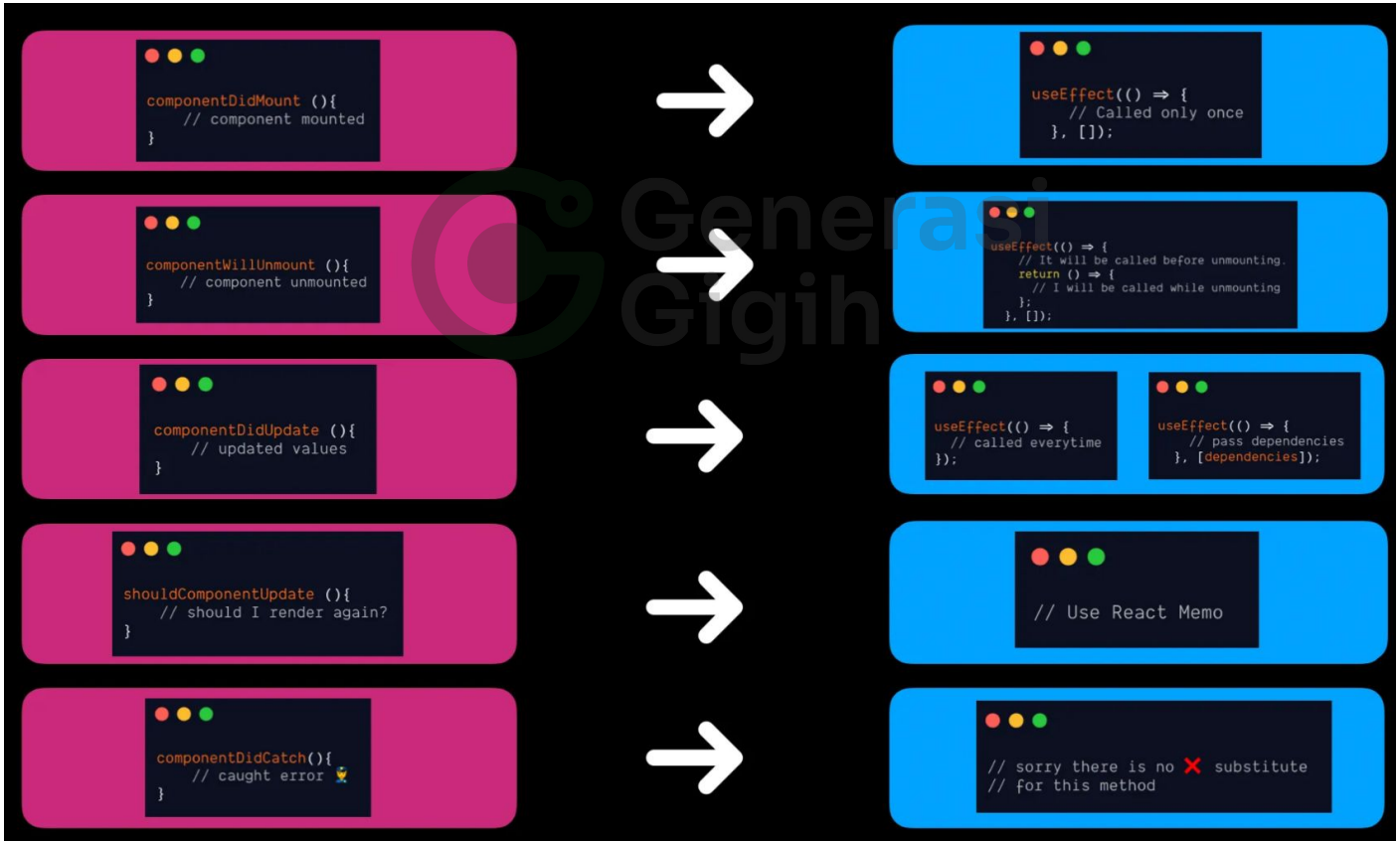
IMPORTANT! [] means we want this to run only once at the beginning

```jsx
import { useEffect, useState } from 'react';

const Sample = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    getUsers();
  }, []);

  const getUsers = async () => {
    const users = await fetch(
      'https://jsonplaceholder.typicode.com/users'
    ).then(response => response.json());

    setUsers(users);
  };

  return (
    <>
      {users.length > 0 && (
        <>
          <div> User list:</div>
          <ul>
            {users.map((user, index) => (
              <li key={index}>
                {user.name} ({user.email})
              </li>
            ))}
          </ul>
        </>
      )}
    </>
  );
};
```

- Hooks equivalently has `useEffect()` API that executes once component is rendered
- Can only be called inside React JSX function, and only on top level (don't call it inside loop)
- Second param is dependency list. Empty array [] meaning it will only run once. You can add dependencies which if changes will make `useEffect()`run again

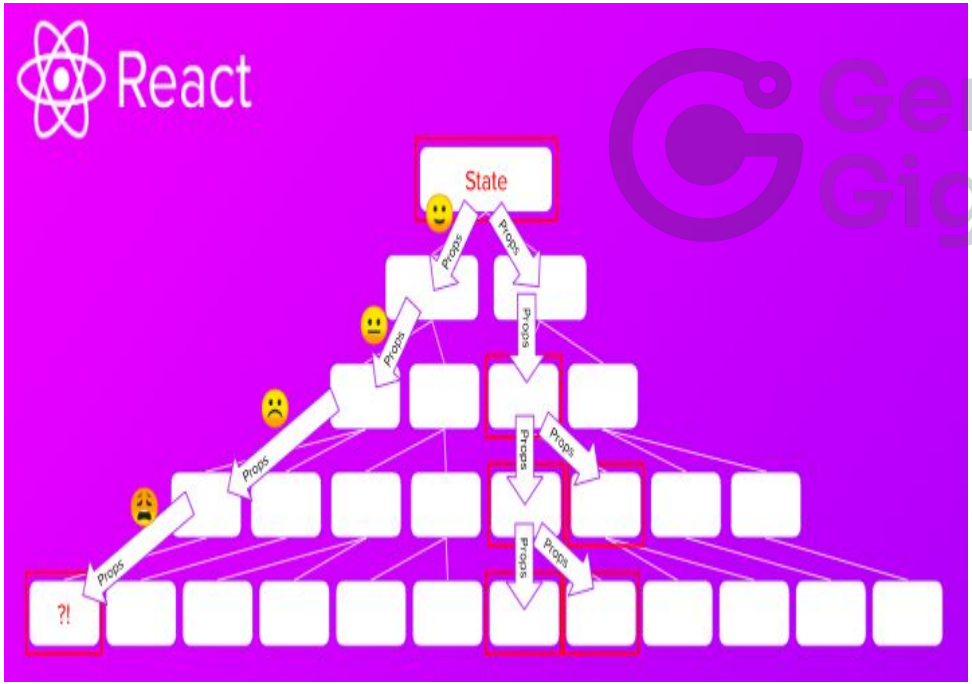
Code

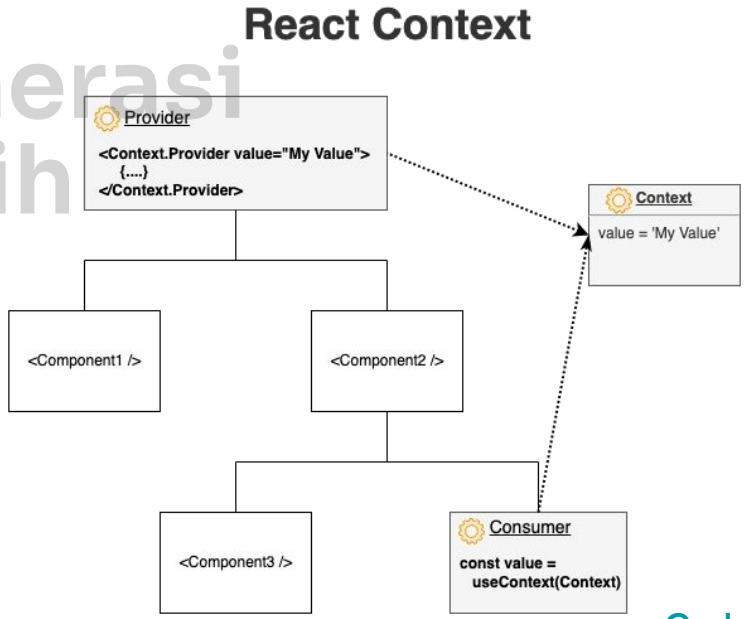
Lifecycle Functional Component



useContext

- Help us from props drilling





Code

Custom Hooks

- Sharing Logic between component

```
const { data, loading, error } = useFetch(
  "https://jsonplaceholder.typicode.com/posts/1"
);
```

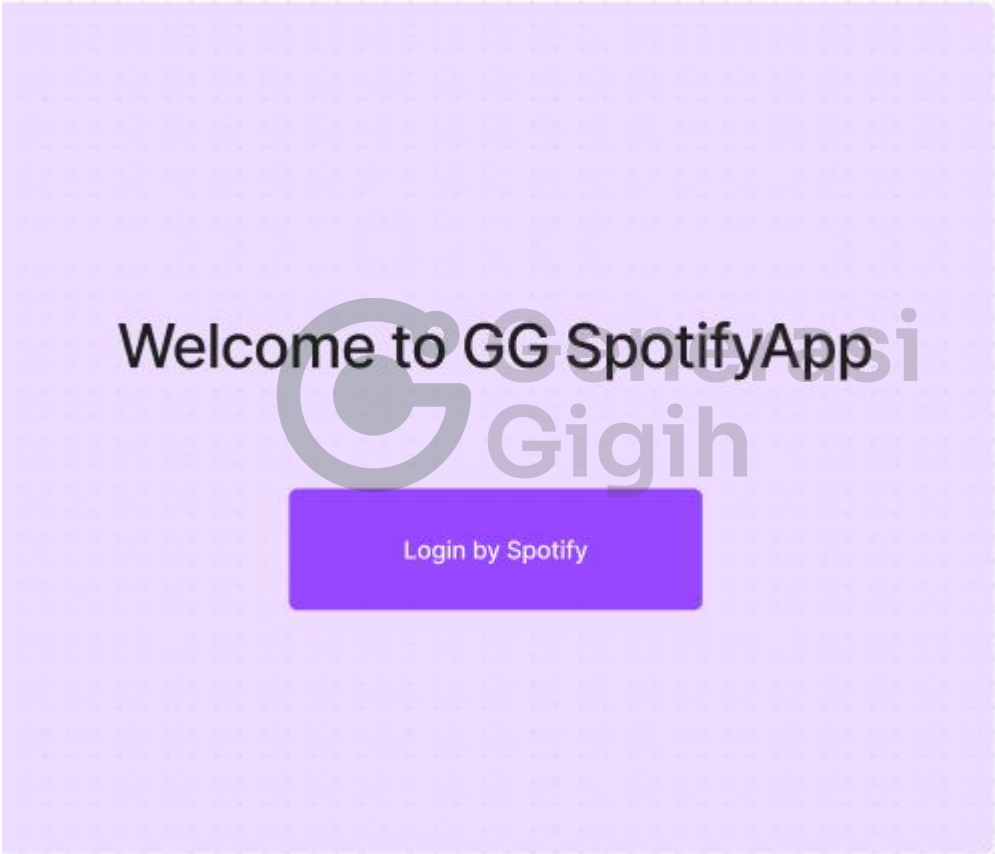
Code

Generasi Gigih

Exercise

Exercise

1. Convert previous exercise from class component to functional component using hooks
2. Connect Spotify Auth API
   - Read it more here in the **Implicit Grant Flow** section
     - +1 if you can use PKCE Flow
   - For the scope, use playlist-modify-private
   - Set up the callback URL as localhost:3000 in the Spotify Dashboard
   - The callback will contains the Access Token, which you'll need for next request. Store that in a state.
3. Create a search song functionalities
   - Have a Search textbox button, when the button is clicked, it calls Spotify Search API. Later on, show the results on the Tracks Table created before.
4. If cannot be finished in class please continue at your own pace outside class (homework)
5. On the next session, some random participants will have to showcase


Welcome to GG SpotifyApp
Login by Spotify

Generasi Gigih

Showcase Time!

Generasi Gigih

Q&A!

Generasi Gigih

Finally, Let’s Wrap Up!

goto

goto impact foundation

Generasi Gigih

See you in the next session!