goto

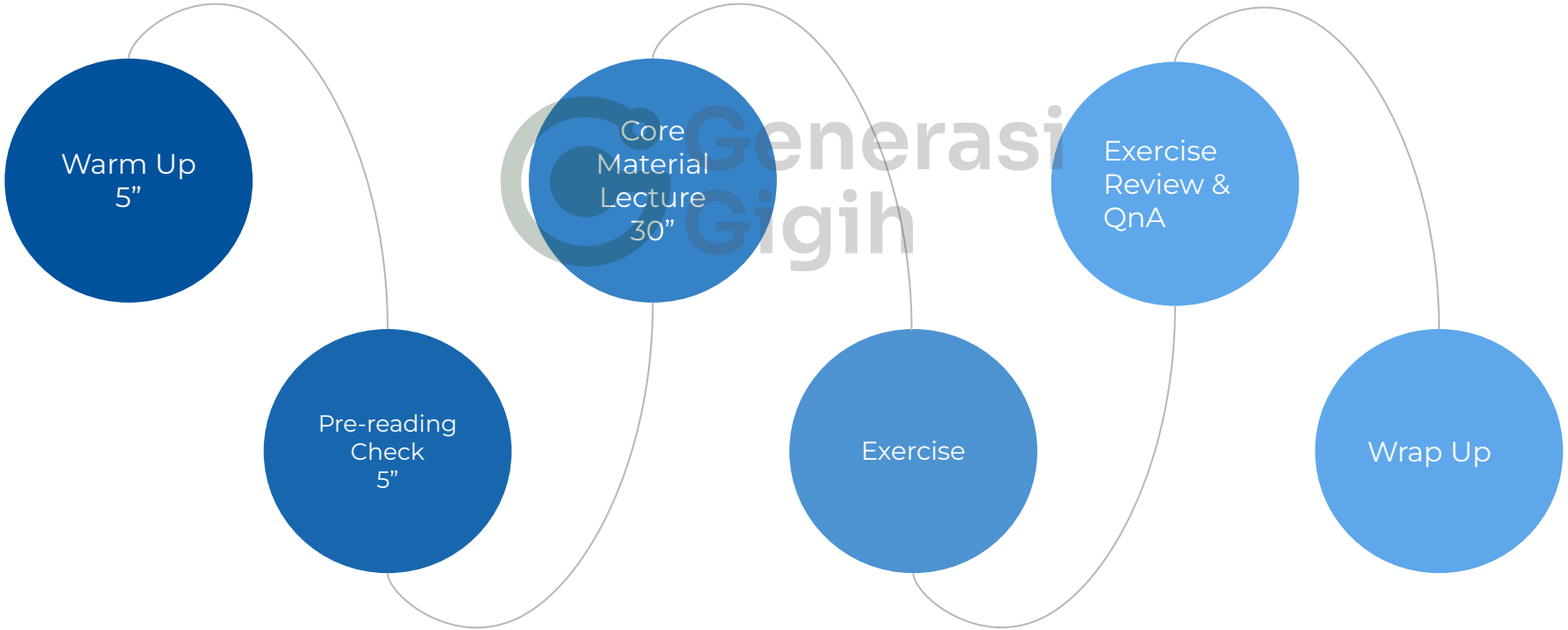
goto impact foundation

Generasi Gigih

Full Stack Engineer

Module 5.2: Forms, Lifting State Up

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 to use Hooks

Generasi Gigih

Showcase Time!

Generasi Gigih

Let’s Discuss

- Form (15”)
  - Entry data form
  - Uncontrolled vs controlled
- Lifting Stateup (15”)
  - Extraction state

Generasi Gigih

Let’s Talk About The Materials

Forms
codesandbox

Forms

- Data entry in user interaction
- Uses
  - HTML form and onSubmit props
  - HTML form elements (input, select, textarea)
  - State
  - onChange & onSubmit event handler

Forms: Example

Important! `<form>` when submitted will refresh our page. We prevent that using `preventDefault()`

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

const sendFormNetworkCall = data => console.log(data);

const Sample = () => {
  const [myText, setMyText] = useState('');

  const handleForm = e => {
    e.preventDefault();
    sendFormNetworkCall(myText);
  };

  const handleMyText = e => setMyText(e.target.value);

  return (
    <>
      <h1>Form</h1>
      <form onSubmit={handleForm}>
        <label htmlFor="mytext">Text:</label>
        <input
          id="mytext"
          type="mytext"
          name="mytext"
          value={myText}
          onChange={handleMyText}
          required
        />
        <button type="submit">Submit</button>
      </form>
    </>
  );
};
```

- `htmlFor` in `<label>` should be the same with id in `<input>`
- value in <input> should refer to myText state
- required in <input> (can only be used inside form)

means if user doesn't input any text, it will show error

`<button>` type `submit` will trigger `<form>`'s `onSubmit`

Forms

- What happens when we have too many inputs?
  - Too many states
  - Too many handlers

Forms: Example

```
const Sample = () => {
  const [myText1, setMyText1] = useState('');
  const [myText2, setMyText2] = useState('');
  const [myText3, setMyText3] = useState('');
  const [myText4, setMyText4] = useState('');

  const handleForm = e => {
    e.preventDefault();
    sendFormNetworkCall({ myText1, myText2, myText3, myText4 });
  };

  const handleMyText1 = e => setMyText1(e.target.value);
  const handleMyText2 = e => setMyText2(e.target.value);
  const handleMyText3 = e => setMyText3(e.target.value);
  const handleMyText4 = e => setMyText4(e.target.value);
```

```
return (
  <>
    <h1>Form</h1>
    <form onSubmit={handleForm}>
      <label htmlFor="mytext1">Text 1:</label>
      <input
        id="mytext1"
        type="text"
        name="myText1"
        value={myText1}
        onChange={handleMyText1}
        required
      />
      <label htmlFor="mytext2">Text 2:</label>
      <input
        id="mytext2"
        type="text"
        name="myText2"
        value={myText2}
        onChange={handleMyText2}
        required
      />
      <label htmlFor="mytext3">Text 3:</label>
      <input
        id="mytext3"
        type="text"
        name="myText3"
        value={myText3}
        onChange={handleMyText3}
        required
      />
      <label htmlFor="mytext4">Text 4:</label>
      <input
        id="mytext4"
        type="text"
        name="myText4"
        value={myText4}
        onChange={handleMyText4}
        required
      />
      <button type="submit">Submit</button>
    </form>
  </>
);
```

Forms

- Solution: combine into one object state & use these JS features
  - Spread operator
  - Computed property names

Forms: Example

Put multiple input states into one form object state

```
const Sample = () => {
  const [myText1, setMyText1] = useState('');
  const [myText2, setMyText2] = useState('');
  const [myText3, setMyText3] = useState('');
  const [myText4, setMyText4] = useState('');

  const handleForm = e => {
    e.preventDefault();
    sendFormNetworkCall({ myText1, myText2, myText3, myText4 });
  };

  const handleMyText1 = e => setMyText1(e.target.value);
  const handleMyText2 = e => setMyText2(e.target.value);
  const handleMyText3 = e => setMyText3(e.target.value);
  const handleMyText4 = e => setMyText4(e.target.value);
```

```
const Sample = () => {
  const [form, setForm] = useState({
    myText1: '',
    myText2: '',
    myText3: '',
    myText4: '',
  });

  const handleForm = e => {
    e.preventDefault();
    sendFormNetworkCall(form);
  };

  const handleMyText = e => {
    const { name, value } = e.target;
    setForm({ ...form, [name]: value });
  };
```

Spread operator `...form` will copy previous values, computed property naming `[name]` will overwrite respective key in state

Forms: Example

```jsx
return (
  <>
    <h1>Form</h1>
    <form onSubmit={handleForm}>
      <label htmlFor="mytext">Text 1:</label>
      <input
        id="mytext"
        type="mytext"
        name="mytext"
        value={myText1}
        onChange={handleMyText1}
        required
      />
      <label htmlFor="mytext">Text 2:</label>
      <input
        id="mytext"
        type="mytext"
        name="mytext"
        value={myText2}
        onChange={handleMyText2}
        required
      />
      <label htmlFor="mytext">Text 3:</label>
      <input
        id="mytext"
        type="mytext"
        name="mytext"
        value={myText3}
        onChange={handleMyText3}
        required
      />
      <label htmlFor="mytext">Text 4:</label>
      <input
        id="mytext"
        type="mytext"
        name="mytext"
        value={myText4}
        onChange={handleMyText4}
        required
      />
      <button type="submit">Submit</button>
    </form>
  </>
);
```

`[name]` (from previous slide will use) will use `name=` attribute here, so make sure it's the same with your state name

```jsx
return (
  <>
    <h1>Form</h1>
    <form onSubmit={handleForm}>
      <label htmlFor="mytext1">Text 1:</label>
      <input
        id="mytext1"
        type="text"
        name="myText1"
        value={form.myText1}
        onChange={handleMyText}
        required
      />
      <label htmlFor="mytext2">Text 2:</label>
      <input
        id="mytext2"
        type="text"
        name="myText2"
        value={form.myText2}
        onChange={handleMyText}
        required
      />
      <label htmlFor="mytext3">Text 3:</label>
      <input
        id="mytext3"
        type="text"
        name="myText3"
        value={form.myText3}
        onChange={handleMyText}
        required
      />
      <label htmlFor="mytext4">Text 4:</label>
      <input
        id="mytext4"
        type="text"
        name="myText4"
        value={form.myText4}
        onChange={handleMyText}
        required
      />
      <button type="submit">Submit</button>
    </form>
  </>
);
```

Generasi Gigih

Controlled VS Uncontrolled

```jsx
function ControlledForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form submitted:', { name, email });
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <label>
        Email:
        <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}

```

```jsx
function UncontrolledForm() {
  const handleSubmit = (event) => {
    event.preventDefault();
    const formData = new FormData(event.target);
    console.log('Form submitted:', Object.fromEntries(formData.entrie
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" name="name" />
      </label>
      <label>
        Email:
        <input type="email" name="email" />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}

```

Controlled VS Uncontrolled

Generasi Gigih

```jsx
function ControlledForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [emailError, setEmailError] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    if (emailError) {
      alert('Please enter a valid email address');
      return;
    }
    console.log('Form submitted:', { name, email });
  }

  const handleEmailChange = (event) => {
    const emailValue = event.target.value;
    setEmail(emailValue);
    setEmailError(emailValue.includes('@') ? '' : 'Invalid email address');
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <label>
        Email:
        <input type="email" value={email} onChange={handleEmailChange} />
        {emailError && <span style={{ color: 'red' }}>{emailError}</span>}
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

```jsx
function UncontrolledForm() {
  const formRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    const emailValue = formRef.current.email.value;
    if (!emailValue.includes('@')) {
      alert('Please enter a valid email address');
      return;
    }
    const formData = new FormData(event.target);
    console.log('Form submitted:', Object.fromEntries(formData.entries
  }

  return (
    <form onSubmit={handleSubmit} ref={formRef}>
      <label>
        Name:
        <input type="text" name="name" />
      </label>
      <label>
        Email:
        <input type="email" name="email" />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Generasi Gigih

Let’s Talk About The Materials

Lifting State Up
codesandbox

Lifting State Up

- Where should our state live?
- Consider the following hierarchy
  - `<UserProfile>`
    - `<ProfileForm>`
- Where should be our User state live? In `<UserProfile>` or `<ProfileForm>`?

Lifting State Up: Before Extracting

```jsx
const UserProfile = () => {
  const [user, setUser] = useState({
    username: '',
    password: '',
    avatarUrl:
      'https://pbs.twimg.com/profile_images/1336281436685541376/fRSl8uJP_400x400.jpg',
  });

  return (
    <div>
      <h1>My Profile</h1>
      <ul>
        <li>
          <div>
            <h2>Avatar</h2>
            <img alt="avatar" src={user.avatarUrl} />
          </div>
        </li>
        <li>
          <ProfileForm />
        </li>
      </ul>
    </div>
  );
};
```

```jsx
return (
  <div>
    <h1>My Profile</h1>
    <ul>
      <li>
        <div>
          <h2>Avatar</h2>
          <img alt="avatar" src={user.avatarUrl} />
        </div>
      </li>
      <li>
        <form onSubmit={handleFormSubmit}>
          <ul>
            <li>
              <label htmlFor="username">Username</label>
              <input
                id="username"
                type="text"
                name="username"
                onChange={handleFormChange}
                value={user.username}
                required
              />
            </li>
            <li>
              <label htmlFor="password">Password</label>
              <input
                id="password"
                type="password"
                name="password"
                onChange={handleFormChange}
                value={user.password}
                required
              />
            </li>
          </ul>

          <button type="submit">Submit</button>
        </form>
      </li>
    </ul>
  </div>
);
```

Lifting State Up: Example

```jsx
const UserProfile = () => {
  const [user, setUser] = useState({
    username: '',
    password: '',
    avatarUrl:
      'https://pbs.twimg.com/profile_images/1336281436685541376/fRSl8uJP_400x400.jpg',
  });

  return (
    <div>
      <h1>My Profile</h1>
      <ul>
        <li>
          <div>
            <h2>Avatar</h2>
            <img alt="avatar" src={user.avatarUrl} />
          </div>
        </li>
        <li>
          <ProfileFormBefore />
        </li>
      </ul>
    </div>
  );
};
```

When we extract to `<ProfileForm>` component, we have 2 source of truths (state). How to keep the state in 1 component?

```jsx
const ProfileForm = () => {
  const [user, setUser] = useState({
    username: '',
    password: '',
    avatarUrl:
      'https://pbs.twimg.com/profile_images/1336281436685541376/fRSl8uJP_400x400.jpg',
  });

  const handleFormChange = e => {
    const { name, value } = e.target;
    setUser({ ...user, [name]: value });
  };

  const handleFormSubmit = e => {
    e.preventDefault();
    console.log(user);
  };

  return (
    <form onSubmit={handleFormSubmit}>
      <ul>
        <li>
          <label htmlFor="username">Username</label>
          <input
            id="username"
            type="text"
            name="username"
            onChange={handleFormChange}
            value={user.username}
            required
          />
        </li>
        <li>
          <label htmlFor="password">Password</label>
          <input
            id="password"
            type="password"
            name="password"
            onChange={handleFormChange}
            value={user.password}
            required
          />
        </li>
      </ul>
      <button type="submit">Submit</button>
```

Lifting State Up: Example

```jsx
const UserProfile = () => {
  const [user, setUser] = useState({
    username: '',
    password: '',
    avatarUrl:
      'https://pbs.twimg.com/profile_images/1336281436685541376/fRSl8uJP_400x400.jpg',
  });

  const handleFormChange = e => {
    const { name, value } = e.target;
    setUser({ ...user, [name]: value });
  };

  const handleFormSubmit = e => {
    e.preventDefault();
    console.log(user);
  };

  return (
    <div>
      <h1>My Profile</h1>
      <ul>
        <li>
          <div>
            <h2>Avatar</h2>
            <img alt="avatar" src={user.avatarUrl} />
          </div>
        </li>
        <li>
          <ProfileForm
            user={user}
            handleFormChange={handleFormChange}
            handleFormSubmit={handleFormSubmit}
          />
        </li>
      </ul>
    </div>
  );
};
```

State should live in parent, and handlers should live with the state

When children needs to access state, we **lift up the state** by using props so parents can pass state (and handlers because handlers and state should be in one level)

```jsx
const ProfileForm = ({ user, handleFormChange, handleFormSubmit }) => (
  <form onSubmit={handleFormSubmit}>
    <ul>
      <li>
        <label htmlFor="username">Username</label>
        <input
          id="username"
          type="text"
          name="username"
          onChange={handleFormChange}
          value={user.username}
          required
        />
      </li>
      <li>
        <label htmlFor="password">Password</label>
        <input
          id="password"
          type="password"
          name="password"
          onChange={handleFormChange}
          value={user.password}
          required
        />
      </li>
    </ul>

    <button type="submit">Submit</button>
  </form>
);
```

Lifting State Up

- There can be a case where our state is required by multiple children of our component
  - `<UserProfile>` does the network call
    - `<ProfileForm>` handles user input
    - `<UserAvatar>` reads user profile and show it as avatar
- In the case above we want to pass `user` state to both children components

Lifting State Up: Example

```jsx
const UserProfile = () => {
  const [user, setUser] = useState({
    username: '',
    password: '',
    avatarUrl: 'https://pbs.twimg.com/profile_images/1336281436685541376/fRSl8uJP_400x400.jpg',
  });

  const handleFormChange = e => {
    const { name, value } = e.target;
    setUser({ ...user, [name]: value });
  };

  const handleFormSubmit = e => {
    e.preventDefault();
    console.log(user);
  };

  return (
    <div>
      <h1>My Profile</h1>
      <ul>
        <li>
          <UserAvatar avatarUrl={user.avatarUrl} />
        </li>
        <li>
          <ProfileForm
            user={user}
            handleFormChange={handleFormChange}
            handleFormSubmit={handleFormSubmit}
          />
        </li>
      </ul>
    </div>
  );
};
```

Extracted UserAvatar component

```jsx
const UserAvatar = ({ avatarUrl }) => (
  <div>
    <h2>Avatar</h2>
    <img alt="avatar" src={avatarUrl} />
  </div>
);
```

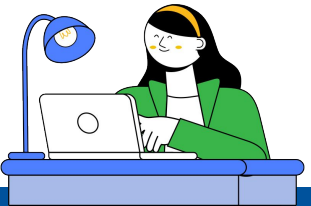
Same as previous slide

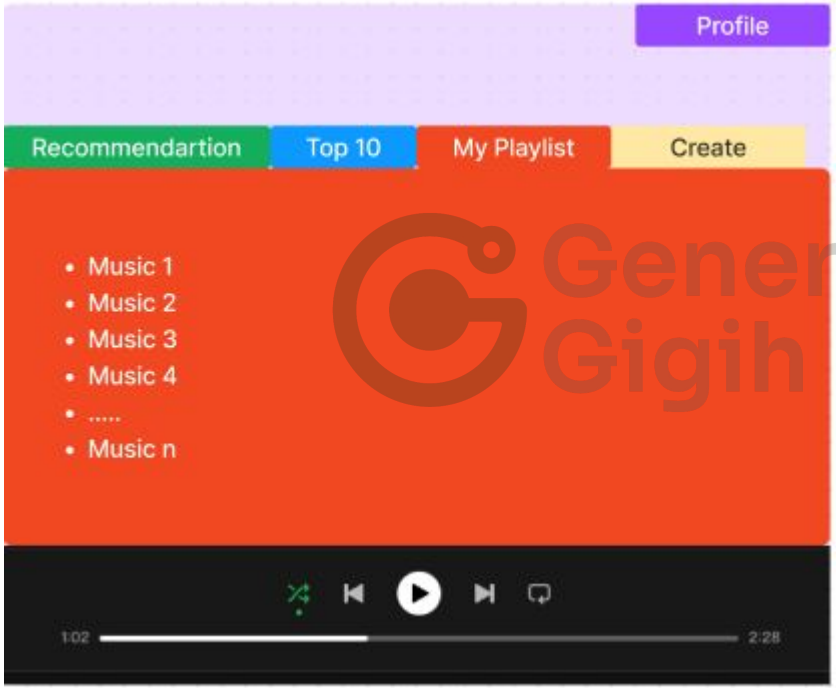
Generasi Gigih

Exercise

Exercise

1. Tabbing/library
    a. recommendation --> Spotify (Read)
    b. My Playlist --> Spotify (CRUD) *nilai plus if can do it
        i. Adding music from search/recommendation
        ii. Can remove music from playlist





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!