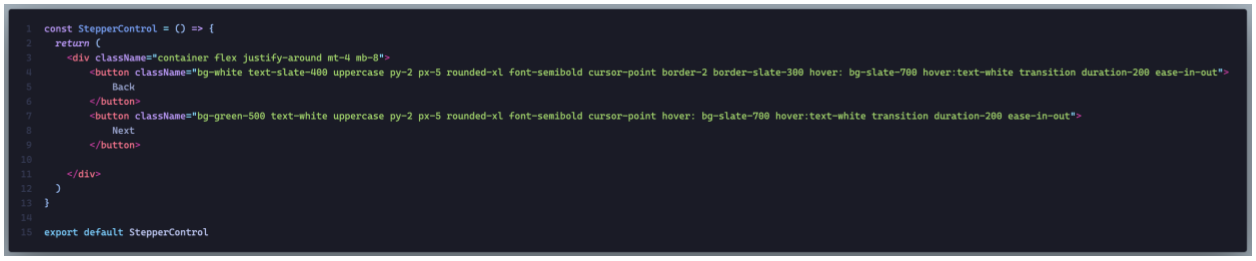
Armand Soto, Samuel Adly, Aiden Schulman
Professor Fried
Internet & Web Technologies
Nov 3. 2024

Project 1 Gray Paper

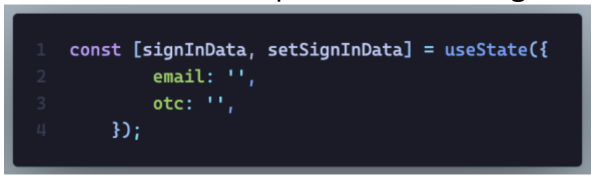
Our project uses Next.js, a web framework that is built on top of React, along with TailwindCSS for styling.

- React is a JavaScript framework used for building user interfaces, primarily focusing on client-side rendering.
- Next.js is a framework that supports server-side rendering, and includes a built-in routing system based on the file system of the project directory.
- TailwindCSS is a CSS framework that allows developers to write CSS code directly in HTML (or JavaScript file in the case of React) by converting predefined class names into CSS.



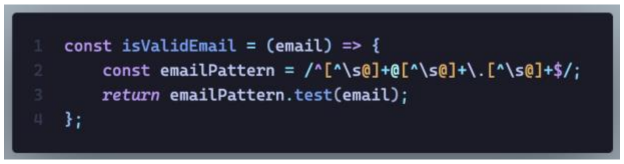
```
const StepperControl = () => {
  return (
    <div className="container flex justify-around mt-4 mb-8">
        <button className="bg-white text-slate-400 uppercase py-2 px-5 rounded-xl font-semibold cursor-point border-2 border-slate-300 hover: bg-slate-700 hover:text-white transition duration-200 ease-in-out">
            Back
        </button>
        <button className="bg-green-500 text-white uppercase py-2 px-5 rounded-xl font-semibold cursor-point hover: bg-slate-700 hover:text-white transition duration-200 ease-in-out">
            Next
        </button>

    </div>
  )
}

export default StepperControl
```

- The above code is an example of a React component. It is stylistically similar to the structure of HTML code, while also allowing for JavaScript logic.
- The element structure is assigned to a JavaScript variable, and is then exported for use in another file.
- Instead of the ‘class’ attribute, React uses the ‘className’ attribute for element classes. The classes in the above component are all recognized by TailwindCSS, and are converted to their respective CSS properties by TailwindCSS e.g., ‘bg-green-500’ is the equivalent to ‘background-color: rgb(34 197 94);’ in CSS.



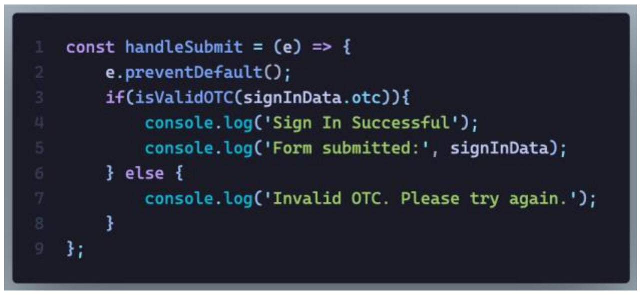
```
const [signInData, setSignInData] = useState({
        email: '',
        otc: '',
    });
```

- Without a useState variable, React components are not re-rendered, even if the value of a variable changes. Therefore, useState must be used when displaying dynamic text in a component, allowing for the webpage to update the new information.

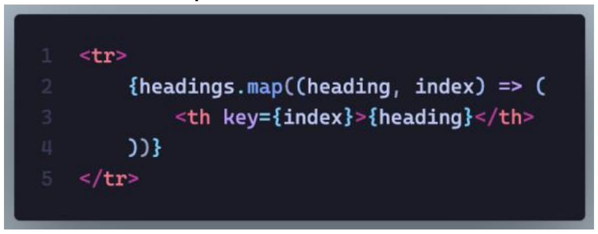


```
const isValidEmail = (email) => {
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailPattern.test(email);
};
```

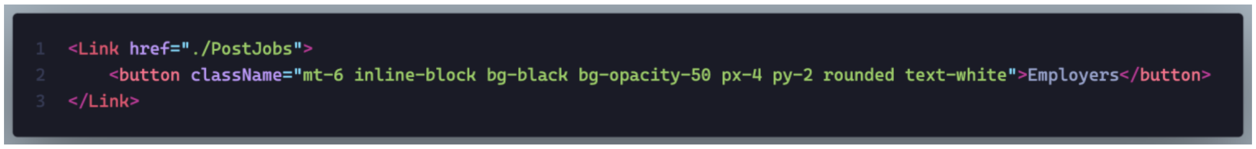
- When the form data is submitted on the sign-in page, the form data is tested to see if it matches the correct format. The above code tests if the text passed to the function matches the email format by comparing the string to a regex pattern.


```
const handleSubmit = (e) => {
    e.preventDefault();
    if(isValidOTC(signInData.otc)){
        console.log('Sign In Successful');
        console.log('Form submitted:', signInData);
    } else {
        console.log('Invalid OTC. Please try again.');
    }
};
```

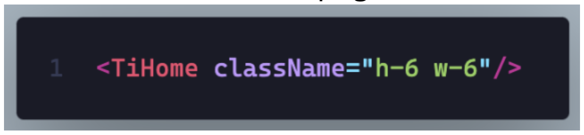
- When submitting form data, the page is usually refreshed when the form is submitted. This behavior is not always preferred, so the handleSubmit function blocks this.
- When the submit event occurs, the preventDefault function is called on it. This prevents the default behavior for the submit even from occurring and allows for the developer to define custom behavior for the event.


```
<tr>
    {headings.map((heading, index) => (
        <th key={index}>{heading}</th>
    ))}
</tr>
```

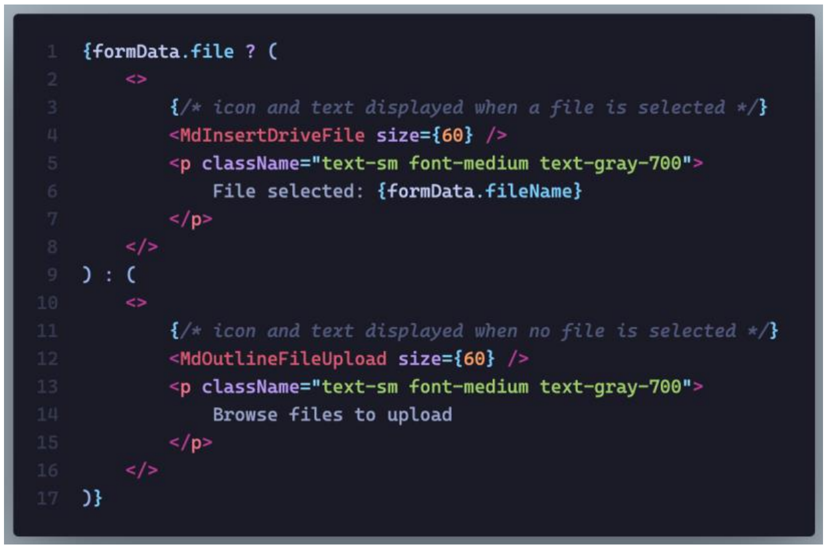
- When creating the footer for the website, we did not want to individually create elements for each of the menu items, as it would result in a lot of repetitive code.
- By storing the menu options in a 2 dimensional array, React allows us to render each element of the array by using the ‘map’ function. We can map each element of the array to a React component which will subsequently be rendered, allowing for more concise code.


```
<Link href="./PostJobs">
    <button className="mt-6 inline-block bg-black bg-opacity-50 px-4 py-2 rounded text-white">Employers</button>
</Link>
```

- In Next.js, the ‘Link’ tag is used when navigating to internal links, or pages within the application. The ‘Link’ tag enables client-side navigation, which is faster and does not cause a full page reload.

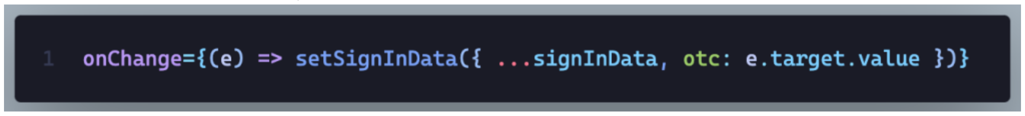

```
<TiHome className="h-6 w-6"/>
```

- Rather than using images, icons from the ‘react-icons’ package are used. These icons offer certain advantages to images, such as no loss of quality when resized, smaller in file size compared to images, and the option to use CSS to style the icon.



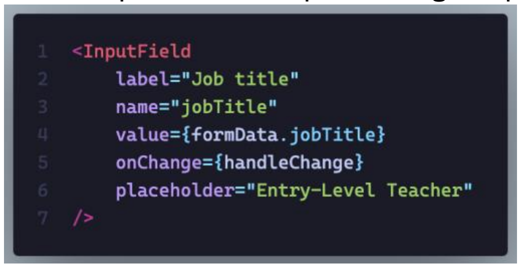
```
{formData.file ? (
    <>
        {/* icon and text displayed when a file is selected */}
        <MdInsertDriveFile size={60} />
        <p className="text-sm font-medium text-gray-700">
            File selected: {formData.fileName}
        </p>
    </>
) : (
    <>
        {/* icon and text displayed when no file is selected */}
        <MdOutlineFileUpload size={60} />
        <p className="text-sm font-medium text-gray-700">
            Browse files to upload
        </p>
    </>
)}
```

- The upload-resume page includes the use of conditional rendering, which is used for changing the icon based on whether a file is uploaded or not. This is accomplished by using a conditional statement: ‘<condition> ? <code if condition is true> : <code if condition is false’.
- The condition tested is if the file property of the form data is defined, and displays a file icon if true, or a browse files icon if false.



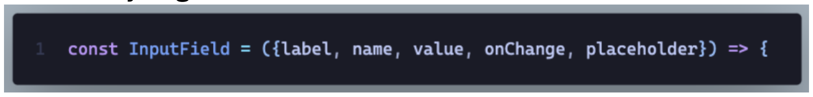
```
onChange={(e) => setSignInData({ ...signInData, otc: e.target.value })}
```

- When the user changes the data in the form, the form does not update automatically. React requires the state variable to be updated using an onChange function so that the variable’s value can be updated whenever the user changes the text in a form.
- By using the spread ‘…’ operator, one specific portion of the form data can be updated while preserving the previous values from other input fields in the form.



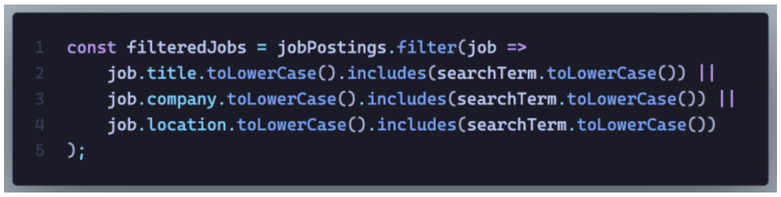
```
<InputField
    label="Job title"
    name="jobTitle"
    value={formData.jobTitle}
    onChange={handleChange}
    placeholder="Entry-Level Teacher"
/>
```

- In order to prevent duplicate code, the InputField component was created to display multiple form inputs with different names and values while using the same styling



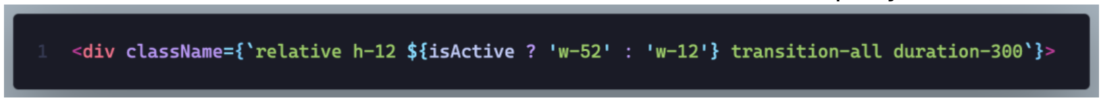
```
const InputField = ({label, name, value, onChange, placeholder}) => {
```

- To have the option to change the label, name, value, etc., the attributes are passed to the component through a de-structured object as the component’s parameter.



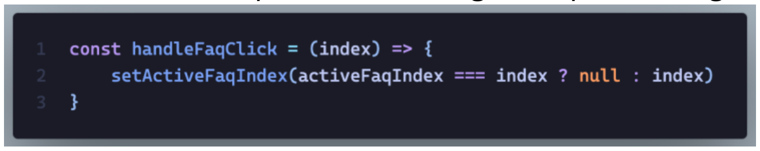
```
const filteredJobs = jobPostings.filter(job =>
    job.title.toLowerCase().includes(searchTerm.toLowerCase()) ||
    job.company.toLowerCase().includes(searchTerm.toLowerCase()) ||
    job.location.toLowerCase().includes(searchTerm.toLowerCase())
);
```

- In order to create a functioning search bar, the displayed elements should match the text in the search bar. To accomplish this, the ‘filter’ method was used.
- The filter method in this code snippet is used to return a list of items where for each item, the passed boolean function returns true. In this case, the item is added to the result list if it includes the search term in its title, company name, or location.



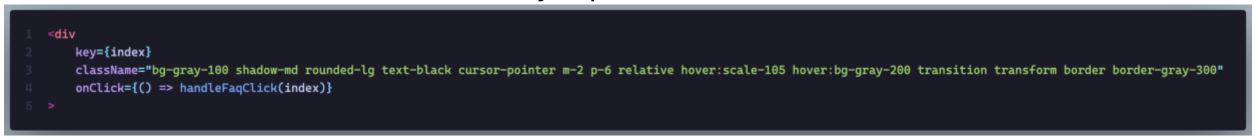
```
<div className={`relative h-12 ${isActive ? 'w-52' : 'w-12'} transition-all duration-300`}>
```

- Along with filtering the correct results, the search bar also expands and contracts when the search icon is clicked. This is done by the use of a dynamic class value, which is implemented using a template string.



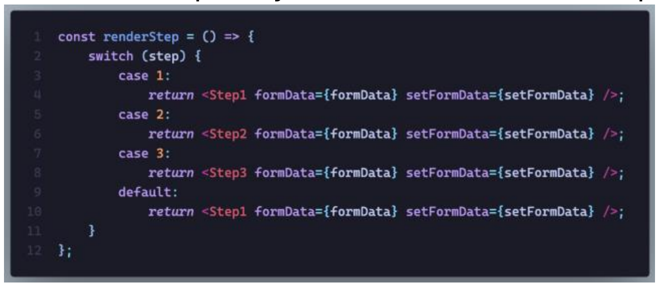
```
const handleFaqClick = (index) => {
    setActiveFaqIndex(activeFaqIndex === index ? null : index)
}
```

- When clicking on an element in the FAQ section of the career-help page, that specific element expands to show the answer to the question. However, including an ‘isExpanded’ useState variable for each element would expand all elements if only one was clicked.
- Therefore, the activeFaqIndex state variable is used instead, where the current value is the index of the currently expanded element in the list of all elements.



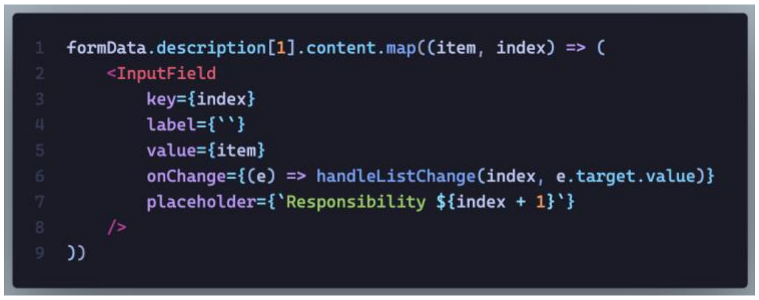
```
<div
    key={index}
    className="bg-gray-100 shadow-md rounded-lg text-black cursor-pointer m-2 p-6 relative hover:scale-105 hover:bg-gray-200 transition transform border border-gray-300"
    onClick={() => handleFaqClick(index)}
>
```

- When the list of FAQ’s are mapped to a div element, the index from the map function is used as the value when calling the handleFaqClick function and subsequently used for the setActiveFaqIndex function call.



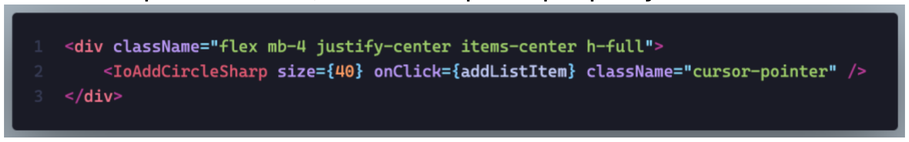
```
const renderStep = () => {
    switch (step) {
        case 1:
            return <Step1 formData={formData} setFormData={setFormData} />;
        case 2:
            return <Step2 formData={formData} setFormData={setFormData} />;
        case 3:
            return <Step3 formData={formData} setFormData={setFormData} />;
        default:
            return <Step1 formData={formData} setFormData={setFormData} />;
    }
};
```

- For the registration page, the progress steps project from the Udemy Course is used. By using React each step component can be separated into their own component, with the formData attribute passed to each step.



```jsx
formData.description[1].content.map((item, index) => (
    <InputField
        key={index}
        label={``}
        value={item}
        onChange={(e) => handleListChange(index, e.target.value)}
        placeholder={`Responsibility ${index + 1}`}
    />
))
```

- In order to add multiple job responsibility fields in the PostJobs page, the map function is once again used. To account for a dynamic number of job responsibilities, the description property is set to be an array.



```jsx
<div className="flex mb-4 justify-center items-center h-full">
    <IoAddCircleSharp size={40} onClick={addListItem} className="cursor-pointer" />
</div>
```

- To give the user control over how many job responsibilities to add, this element was created to add a new blank input field whenever it receives a click event.
- The click event executes the addListItem function, which pushes the blank input field to the back of the description list.