FSD Laboratory 05

Arin Khopkar, TY B.Tech CSE, Panel I, I2, 62, 1032221073

Aim: Design and develop an interactive user interface using React.

Objectives:
1. Articulate what React is and why it is useful.
2. Explore the basic architecture of a React application.
3. Use React components to build interactive interfaces

Theory:
1. What is React? Steps to run React app using create-react-app.

React is a popular JavaScript library for building user interfaces, especially single-page applications. It is developed and maintained by Facebook, and it allows developers to create reusable UI components that efficiently update and render data as the state changes.

Key Features of React:

- **Component-Based Architecture**: React applications are composed of reusable components, which help in organizing UI development and reusing code.
- **Virtual DOM**: React uses a virtual representation of the DOM to efficiently update the UI by minimizing direct manipulation of the actual DOM, resulting in faster rendering.
- **Unidirectional Data Flow**: Data in React flows from parent to child components through props, making data management predictable and easier to debug.
- **JSX (JavaScript XML)**: React uses JSX, which allows you to write HTML-like code directly in JavaScript, simplifying the creation of complex UI components.
- **State Management**: Components can manage internal states, and the UI updates automatically when the state changes.
- **Ecosystem**: React has a rich ecosystem of libraries for routing, state management (e.g., Redux), form handling, and more.

Steps to Run a React App Using `create-react-app`:

create-react-app is a command-line tool that sets up a new React application with a well-structured environment, including Webpack, Babel, and other essential tools.

1. Install Node.js:

Before creating a React app, ensure you have Node.js and npm (Node Package Manager) installed.

Check if Node.js is installed:

```
node -v
```

```
npm -v
```

2. Install create-react-app globally (optional):

This step is no longer required as you can use npx (which comes with npm) to create a React app, but you can still install create-react-app globally if preferred.

```
npm install -g create-react-app
```

3. Create a New React App:

Use npx to create a new React app in a directory. This command will automatically create a new project with all dependencies.

```
npx create-react-app my-app
```

This command will create a new folder named `my-app` and install all the required dependencies for the React application.

4. Navigate to the Project Directory:

Move into the project folder to start working on your app.

```
cd my-app
```

5. Run the Development Server:

Once inside the project directory, you can start the React app using the following command. This will run the app in development mode, and it will be accessible in the browser at http://localhost:3000.

```
npm start
```

6. Build the App for Production (optional):

If you're ready to deploy your app, you can build an optimized production version using:

```
npm run build
```

This will create an optimized build of your app in the `build/` directory.

7. Open the React App:

After running npm start, your default browser should open with http://localhost:3000, displaying your new React app.

2. Passing data through props (Small example)

Props (short for "properties") are a mechanism for passing data from a parent component to a child component in React. Props allow you to make components dynamic and reusable by giving them the ability to receive and display different data.

Example of Passing Data Using Props:

```
// ParentComponent.js
import React from "react";
import ChildComponent from "./ChildComponent";

function ParentComponent() {
  const userName = "John Doe";
  return (
    <div>
      <h1>Parent Component</h1>
      {/* Passing the userName variable to the child component as a prop */}
      <ChildComponent name={userName} />
    </div>
  );
}

export default ParentComponent;

// ChildComponent.js
import React from "react";

// Receiving props in the child component
function ChildComponent(props) {
  return (
    <div>
      <h2>Child Component</h2>
      <p>Hello, {props.name}!</p> {/* Accessing the name prop */}
    </div>
  );
}

export default ChildComponent;
```

Explanation:

- **ParentComponent**: The ParentComponent defines a userName variable and passes it to the ChildComponent using the name prop (<ChildComponent name={userName} />).
- **ChildComponent**: The ChildComponent receives props as a parameter and accesses the name prop using props.name. This allows the child component to display the name passed from the parent.

Output:

Parent Component

Child Component

Hello, John Doe!

This is how data flows from the parent component (ParentComponent) to the child component (ChildComponent) via props.

FAQ:
1. What are React states and hooks?

A1) In React, **state** refers to an object that stores dynamic data for a component and influences how the component behaves and renders. State allows components to manage and react to changes, making the component dynamic and interactive.

- **State in a Class Component**: In class-based components, this.state holds the component's state, and you can update it using this.setState().
- **State in a Functional Component**: Functional components originally lacked state, but with the introduction of **React Hooks** (e.g., useState), functional components can now manage state as well.

Key Characteristics of React State:

- **Mutable**: Unlike props, state is internal to the component and can be changed (mutated).
- **Triggers Re-Rendering**: When the state is updated, React automatically re-renders the component to reflect the new state.
- **Private to Component**: State is encapsulated within the component that owns it and cannot be directly accessed by child components unless passed via props.

React Hooks are special functions introduced in React 16.8 that allow developers to use state and other React features (such as lifecycle methods) in **functional components**, which were previously only available in class components.

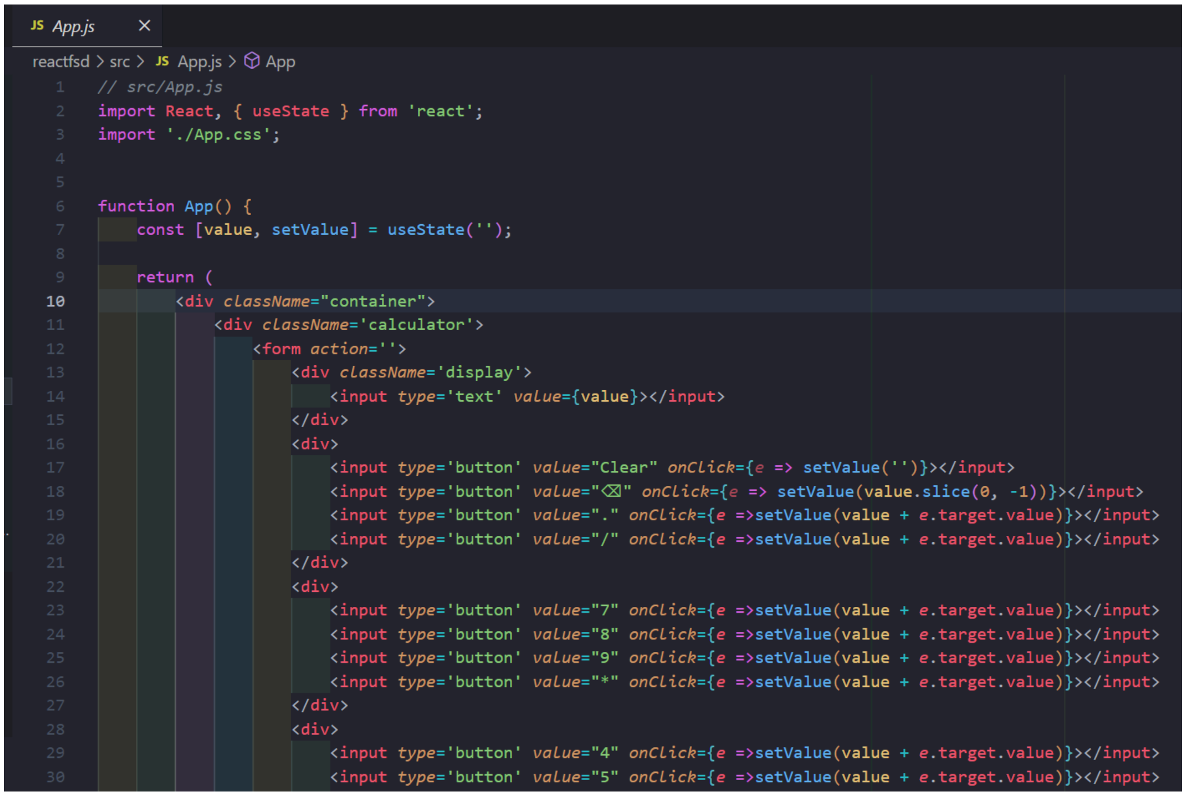
Key Features of Hooks:

- **useState**: Hook for managing state in functional components.
- **useEffect**: Hook for handling side effects (e.g., data fetching, subscriptions, etc.).
- **useContext**: Hook for using context to manage global state.
- **useReducer**: Hook for managing complex state logic (similar to Redux).
- **Custom Hooks**: You can create your own custom hooks to encapsulate and reuse logic.

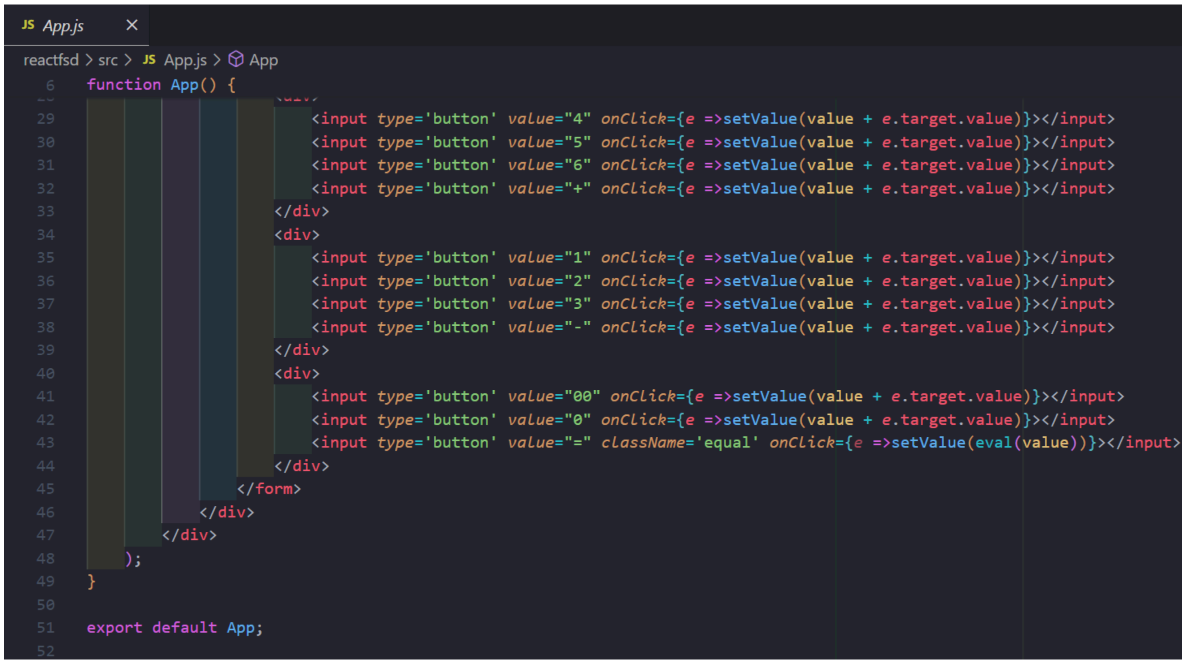
Hooks offer a way to reuse logic across components (custom hooks), allow functional components to manage state and lifecycle events, and reduce the complexity associated with managing large class components.

Code:

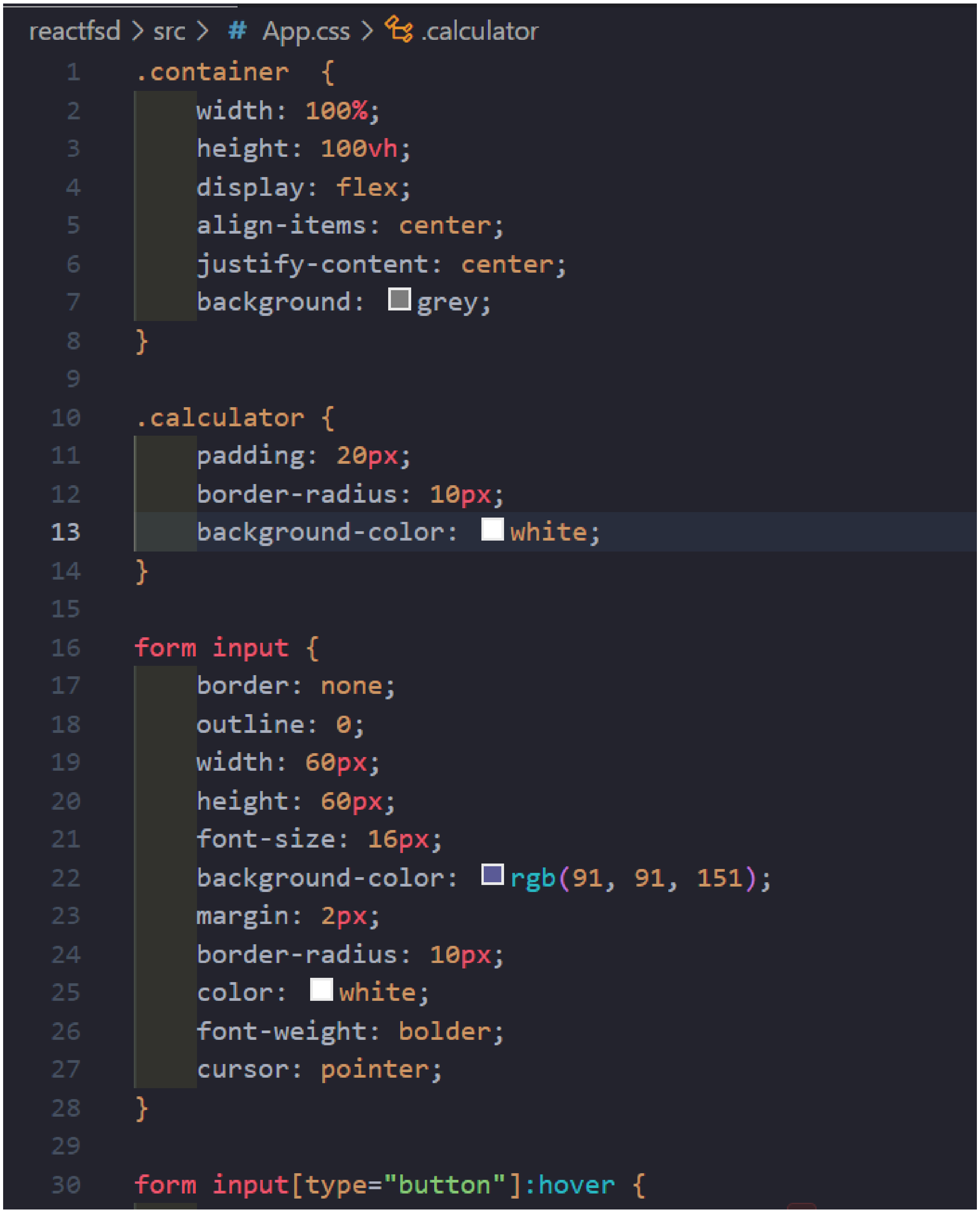
UI for calculator using React: (App.js, App.css)



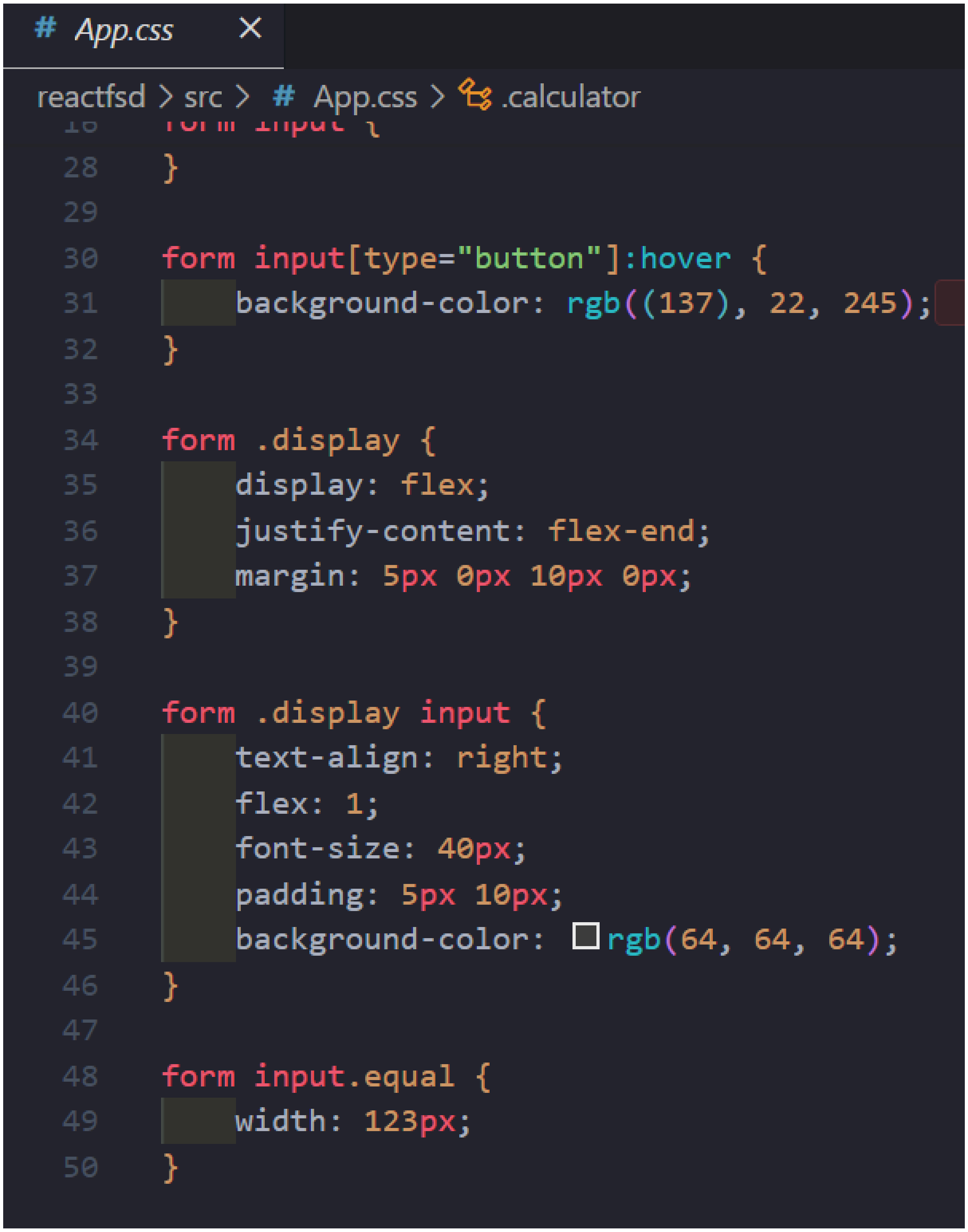
```
// src/App.js
import React, { useState } from 'react';
import './App.css';


function App() {
  const [value, setValue] = useState('');

  return (
    <div className="container">
      <div className='calculator'>
        <form action=''>
          <div className='display'>
            <input type='text' value={value}></input>
          </div>
          <div>
            <input type='button' value="Clear" onClick={e => setValue('')}></input>
            <input type='button' value="⌫" onClick={e => setValue(value.slice(0, -1))}></input>
            <input type='button' value="." onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="/" onClick={e =>setValue(value + e.target.value)}></input>
          </div>
          <div>
            <input type='button' value="7" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="8" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="9" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="*" onClick={e =>setValue(value + e.target.value)}></input>
          </div>
          <div>
            <input type='button' value="4" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="5" onClick={e =>setValue(value + e.target.value)}></input>
```



```
function App() {
            <input type='button' value="4" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="5" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="6" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="+" onClick={e =>setValue(value + e.target.value)}></input>
          </div>
          <div>
            <input type='button' value="1" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="2" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="3" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="-" onClick={e =>setValue(value + e.target.value)}></input>
          </div>
          <div>
            <input type='button' value="00" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="0" onClick={e =>setValue(value + e.target.value)}></input>
            <input type='button' value="=" className='equal' onClick={e =>setValue(eval(value))}></input>
          </div>
        </form>
      </div>
    </div>
  );
}

export default App;
```

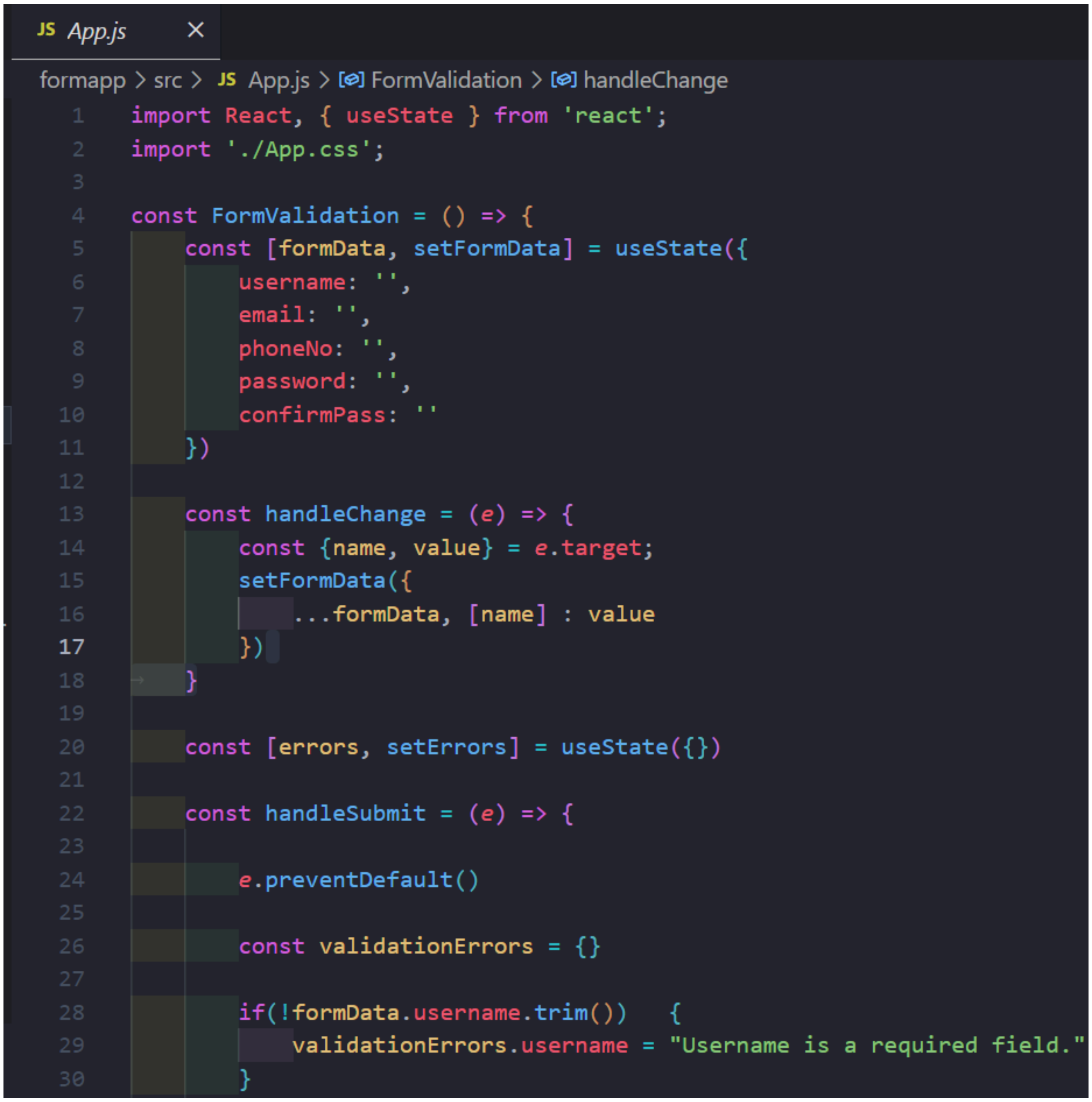

reactfsd > src > # App.css > .calculator

```
.container  {
    width: 100%;
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    background: grey;
}

.calculator {
    padding: 20px;
    border-radius: 10px;
    background-color: white;
}

form input {
    border: none;
    outline: 0;
    width: 60px;
    height: 60px;
    font-size: 16px;
    background-color: rgb(91, 91, 151);
    margin: 2px;
    border-radius: 10px;
    color: white;
    font-weight: bolder;
    cursor: pointer;
}

form input[type="button"]:hover {
```

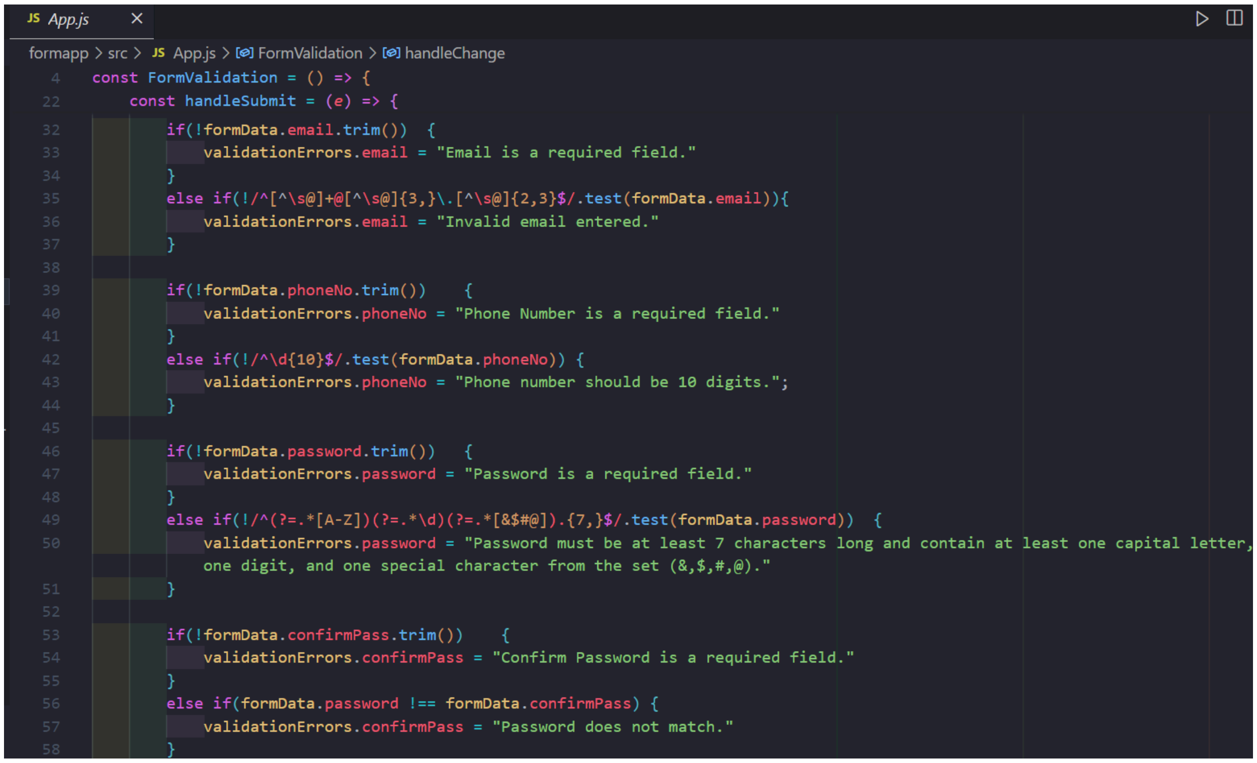

App.css
reactfsd > src > App.css > .calculator

```css
}

form input[type="button"]:hover {
    background-color: rgb((137), 22, 245);
}

form .display {
    display: flex;
    justify-content: flex-end;
    margin: 5px 0px 10px 0px;
}

form .display input {
    text-align: right;
    flex: 1;
    font-size: 40px;
    padding: 5px 10px;
    background-color: rgb(64, 64, 64);
}

form input.equal {
    width: 123px;
}
```

2.)UI for form to gather individual's data using React:

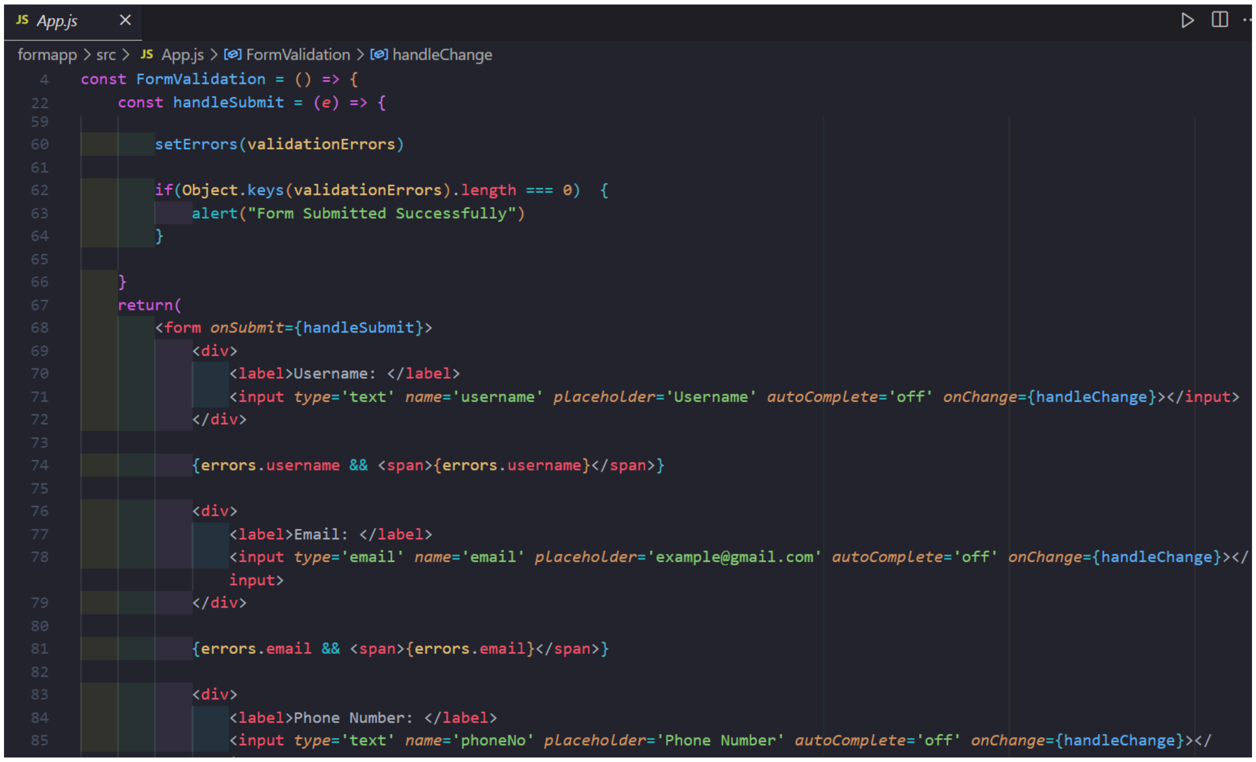

App.js
formapp > src > App.js > FormValidation > handleChange

```
import React, { useState } from 'react';
import './App.css';

const FormValidation = () => {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        phoneNo: '',
        password: '',
        confirmPass: ''
    })

    const handleChange = (e) => {
        const {name, value} = e.target;
        setFormData({
            ...formData, [name] : value
        })
    }

    const [errors, setErrors] = useState({})

    const handleSubmit = (e) => {

        e.preventDefault()

        const validationErrors = {}

        if(!formData.username.trim())   {
            validationErrors.username = "Username is a required field."
        }
```


App.js

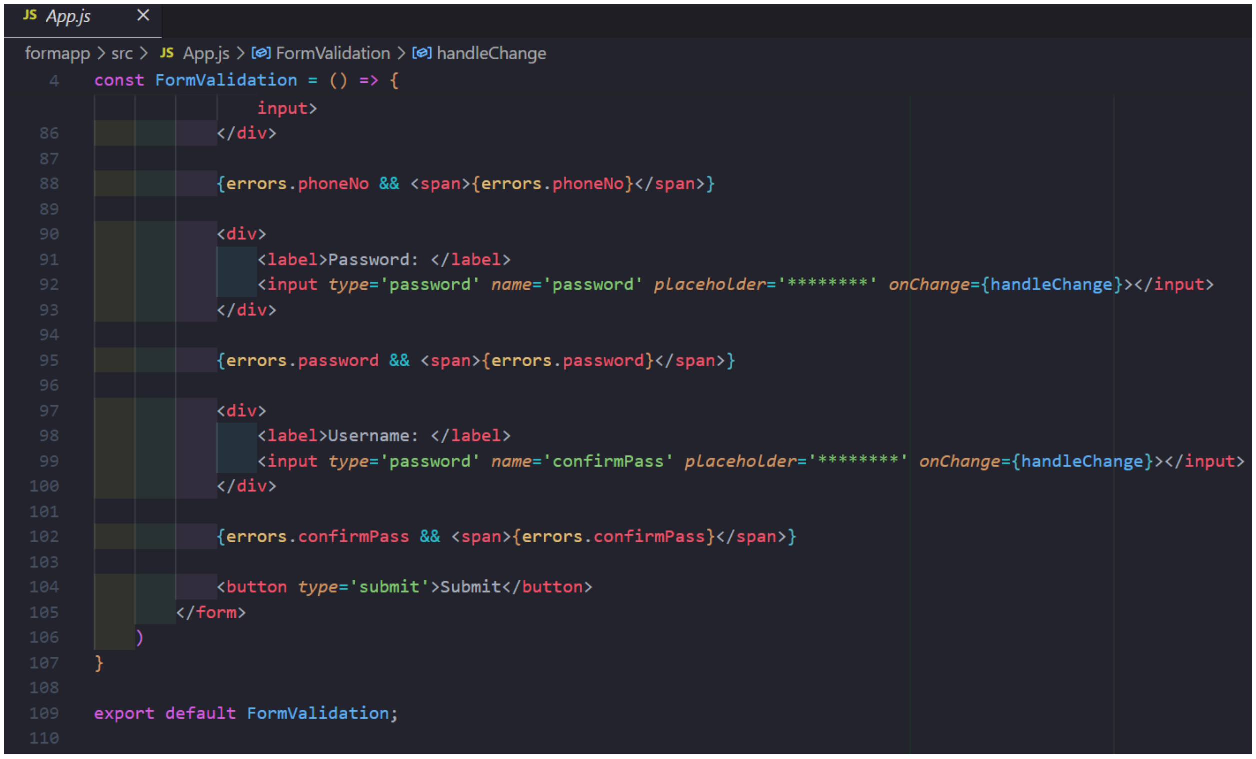
formapp > src > App.js > FormValidation > handleChange

```
const FormValidation = () => {
    const handleSubmit = (e) => {
        if(!formData.email.trim())  {
            validationErrors.email = "Email is a required field."
        }
        else if(!/^[^\s@]+@[^\s@]{3,}\.[^\s@]{2,3}$/.test(formData.email)){
            validationErrors.email = "Invalid email entered."
        }

        if(!formData.phoneNo.trim())    {
            validationErrors.phoneNo = "Phone Number is a required field."
        }
        else if(!/^\d{10}$/.test(formData.phoneNo)) {
            validationErrors.phoneNo = "Phone number should be 10 digits.";
        }

        if(!formData.password.trim())   {
            validationErrors.password = "Password is a required field."
        }
        else if(!/^(?=.*[A-Z])(?=.*\d)(?=.*[&$#@]).{7,}$/.test(formData.password))  {
            validationErrors.password = "Password must be at least 7 characters long and contain at least one capital letter,
            one digit, and one special character from the set (&,$,#,@)."
        }

        if(!formData.confirmPass.trim())    {
            validationErrors.confirmPass = "Confirm Password is a required field."
        }
        else if(formData.password !== formData.confirmPass) {
            validationErrors.confirmPass = "Password does not match."
        }
```


App.js

formapp > src > App.js > FormValidation > handleChange

```
const FormValidation = () => {
    const handleSubmit = (e) => {

        setErrors(validationErrors)

        if(Object.keys(validationErrors).length === 0)  {
            alert("Form Submitted Successfully")
        }

    }
    return(
        <form onSubmit={handleSubmit}>
            <div>
                <label>Username: </label>
                <input type='text' name='username' placeholder='Username' autoComplete='off' onChange={handleChange}></input>
            </div>

            {errors.username && <span>{errors.username}</span>}

            <div>
                <label>Email: </label>
                <input type='email' name='email' placeholder='example@gmail.com' autoComplete='off' onChange={handleChange}></
                input>
            </div>

            {errors.email && <span>{errors.email}</span>}

            <div>
                <label>Phone Number: </label>
                <input type='text' name='phoneNo' placeholder='Phone Number' autoComplete='off' onChange={handleChange}></
```



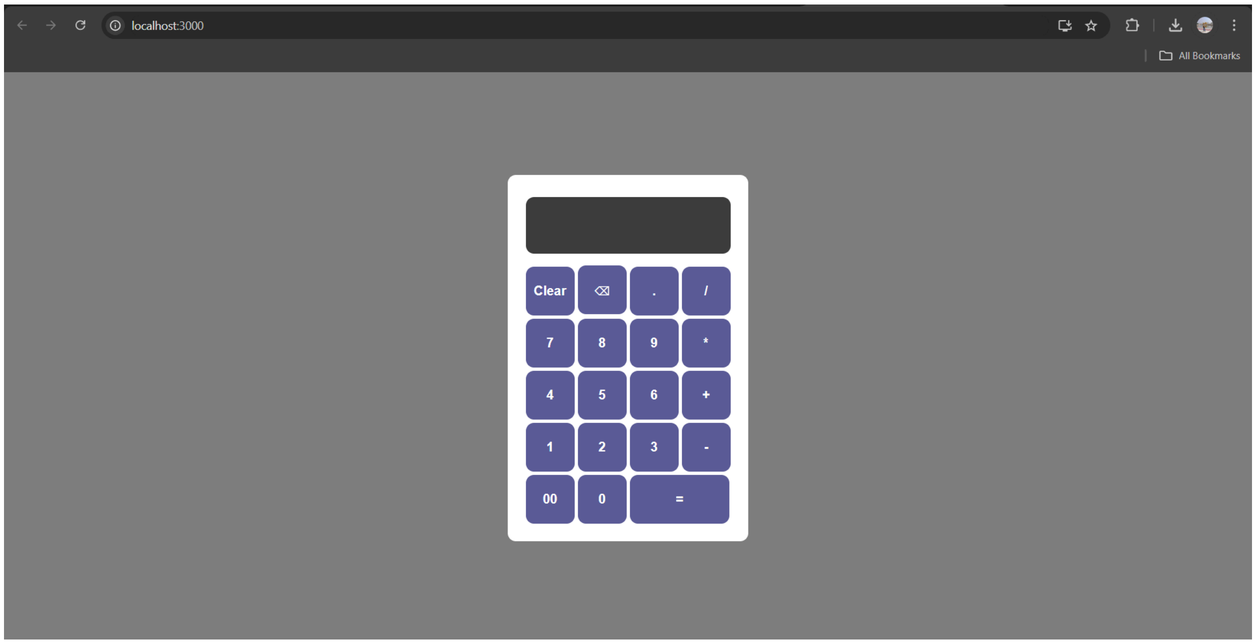
App.js

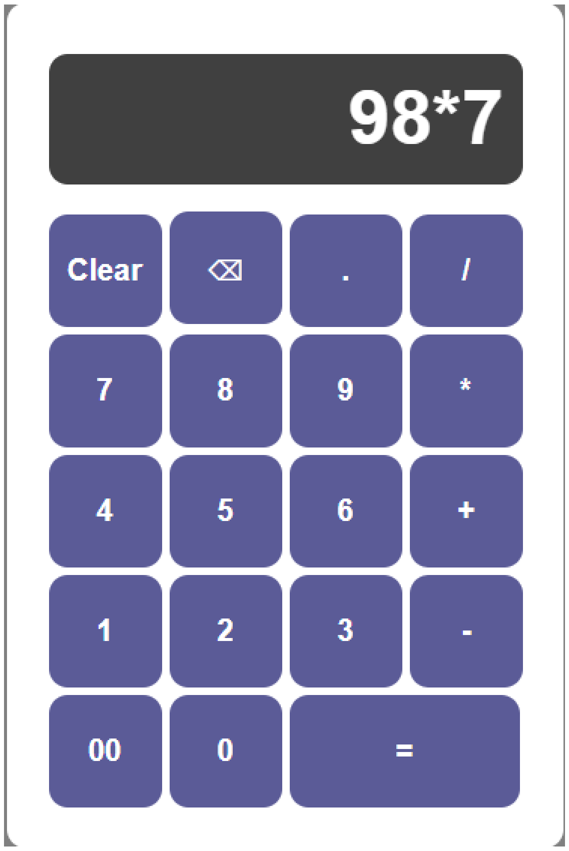
formapp > src > App.js > FormValidation > handleChange

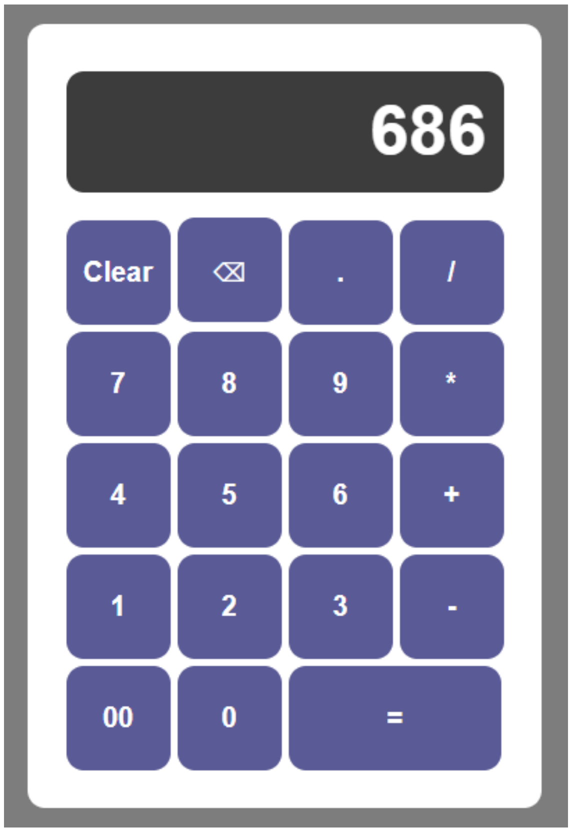
```
const FormValidation = () => {
                input>
            </div>

            {errors.phoneNo && <span>{errors.phoneNo}</span>}

            <div>
                <label>Password: </label>
                <input type='password' name='password' placeholder='********' onChange={handleChange}></input>
            </div>

            {errors.password && <span>{errors.password}</span>}

            <div>
                <label>Username: </label>
                <input type='password' name='confirmPass' placeholder='********' onChange={handleChange}></input>
            </div>

            {errors.confirmPass && <span>{errors.confirmPass}</span>}

            <button type='submit'>Submit</button>
        </form>
    )
}

export default FormValidation;
```

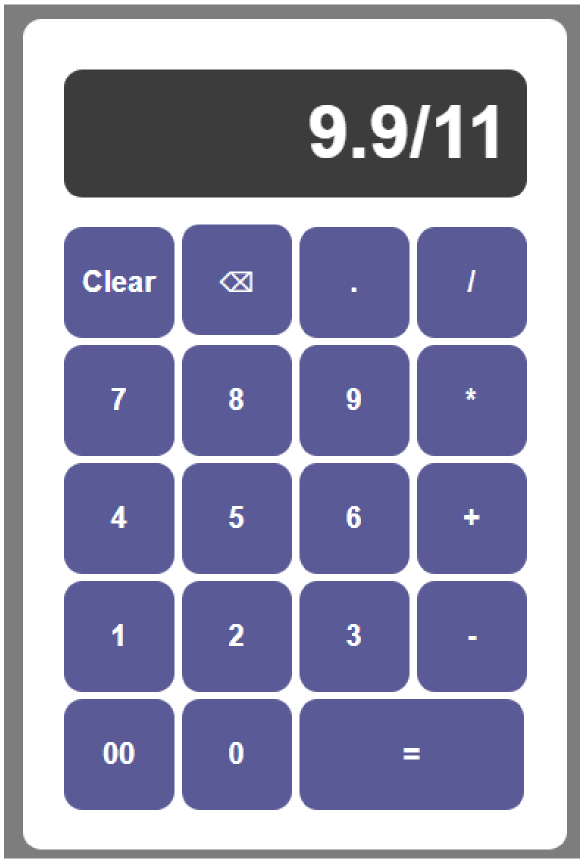
Output: Screenshots of the output to be attached.

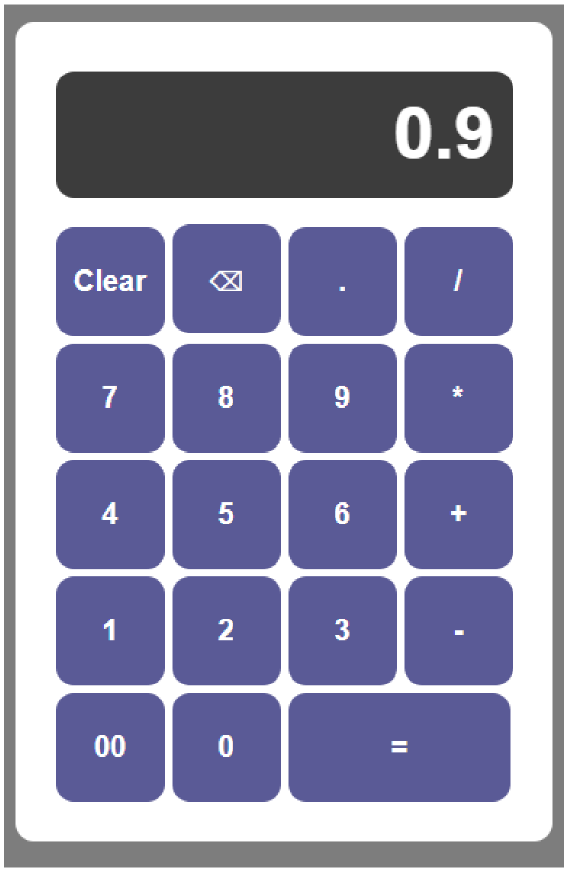
1.) **UI for calculator using REACT:**

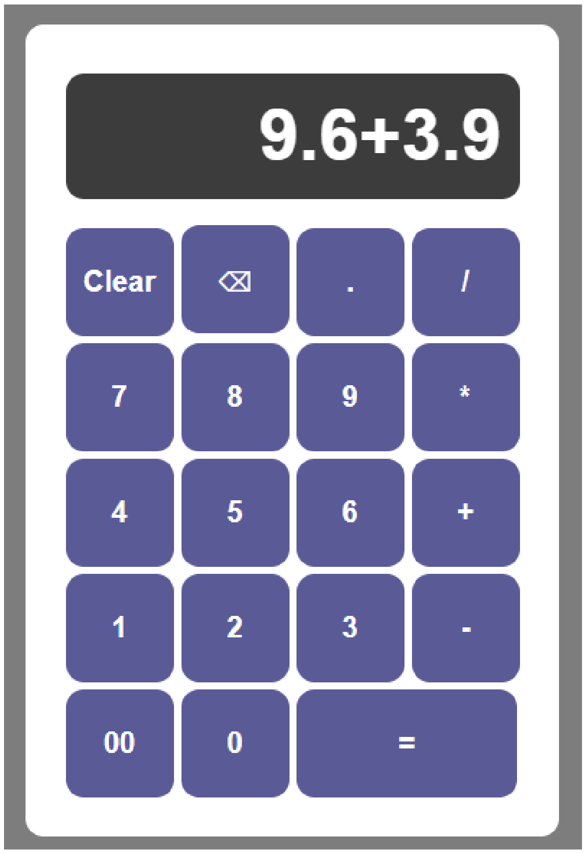


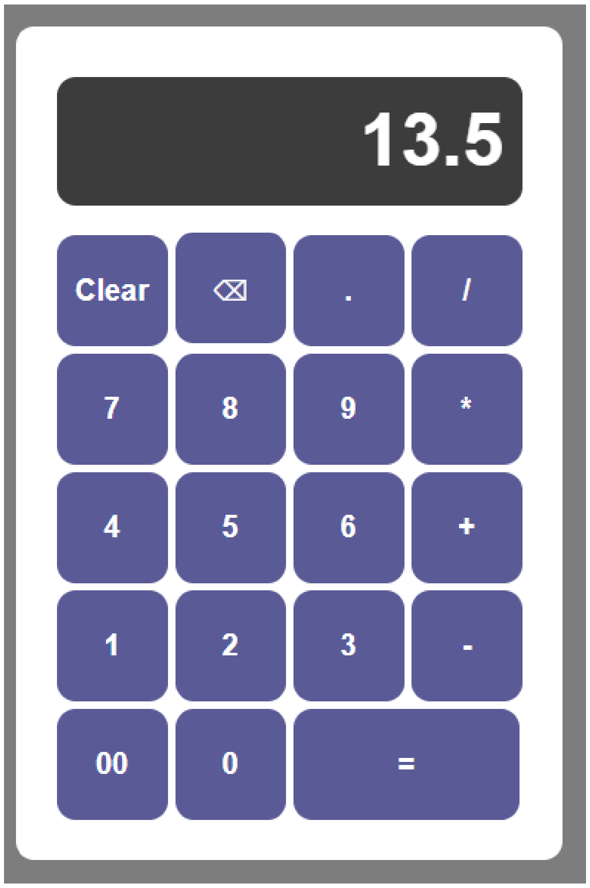

98*7
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=

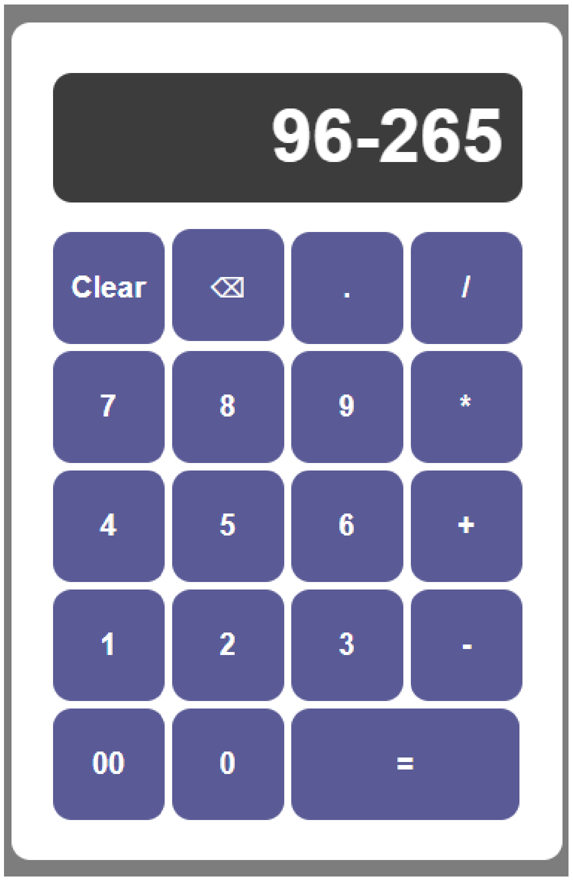

686
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=

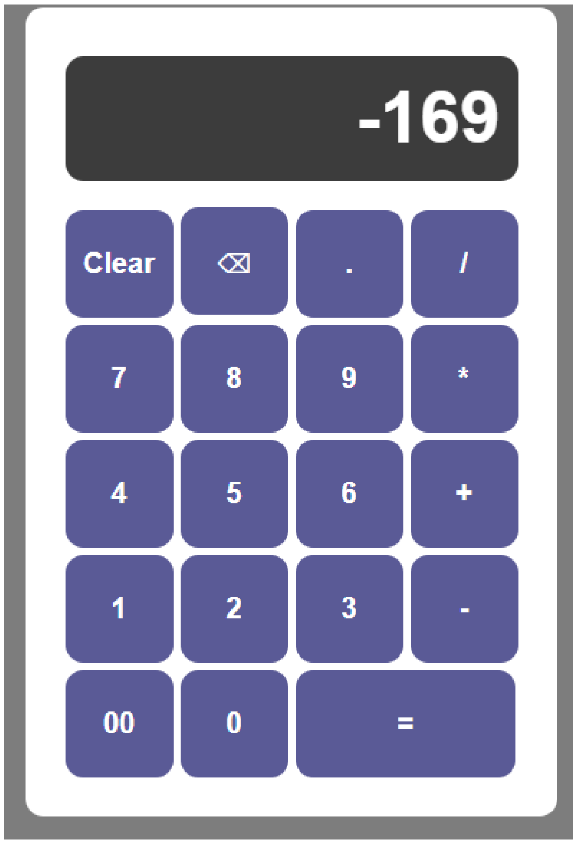

9.9/11
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=


0.9
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=

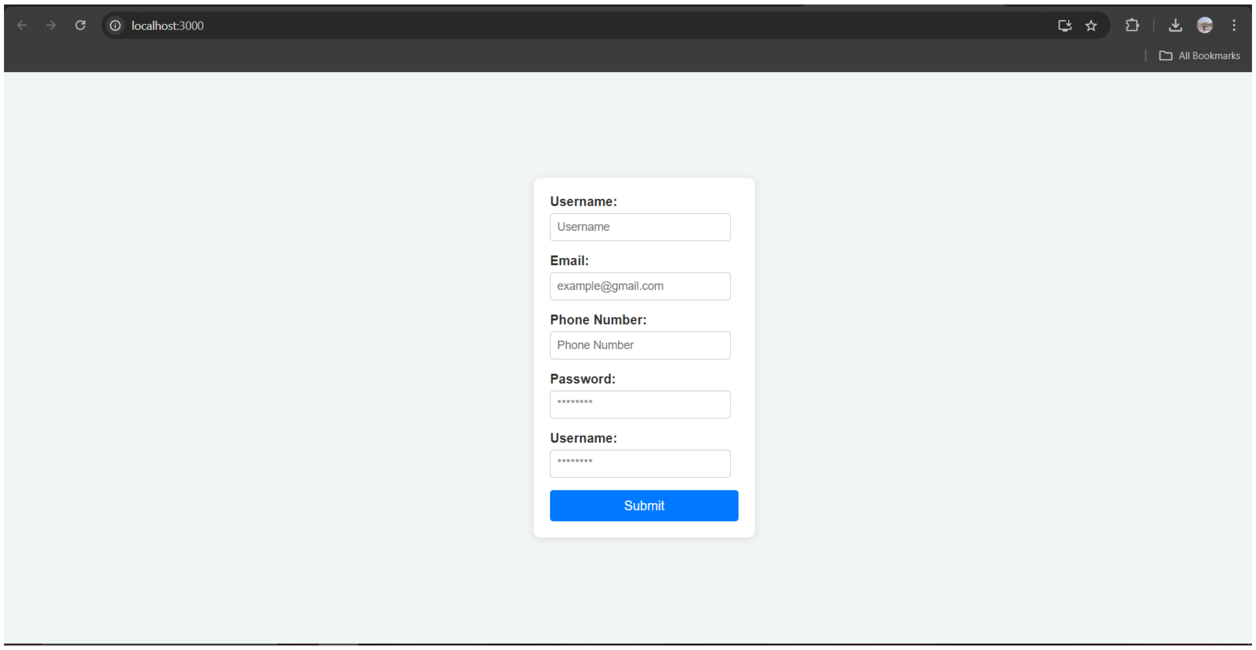

9.6+3.9
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=

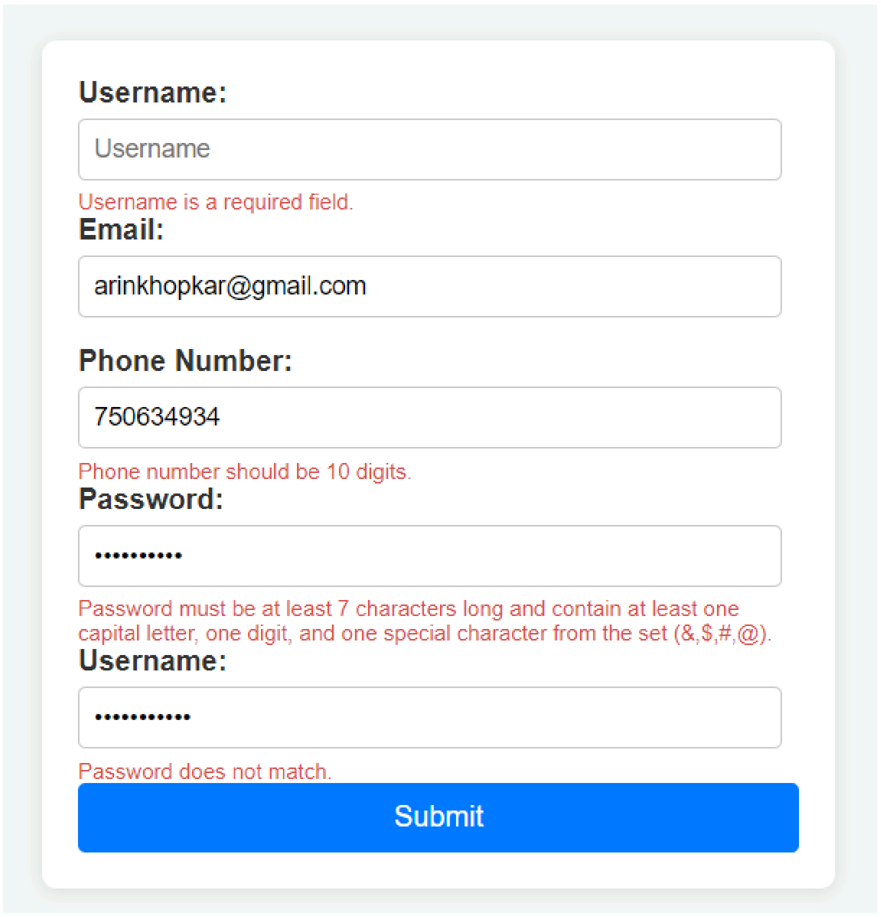

13.5
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=


96-265
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=


-169
Clear
⌫
.
/
7
8
9
*
4
5
6
+
1
2
3
-
00
0
=

2.)UI for Form to gather an individual's data using React:




Username:

Username

Username is a required field.

Email:

arinkhopkar@gmail.com

Phone Number:

750634934

Phone number should be 10 digits.

Password:

••••••••••

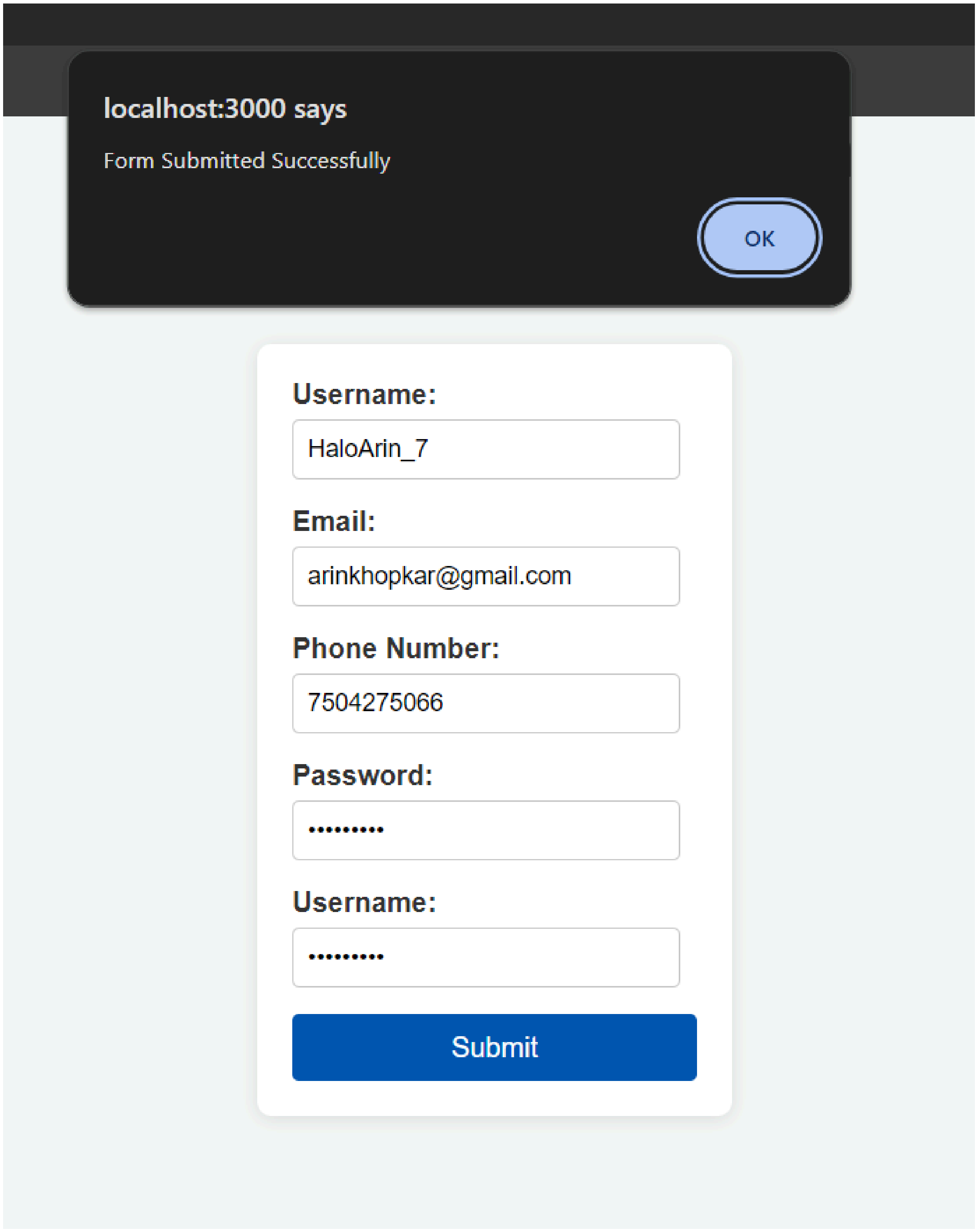
Password must be at least 7 characters long and contain at least one capital letter, one digit, and one special character from the set (&,$,#,@).

Username:

•••••••••••

Password does not match.

Submit



localhost:3000 says

Form Submitted Successfully

Username:	HaloArin_7
Email:	arinkhopkar@gmail.com
Phone Number:	7504275066
Password:	•••••••••
Username:	•••••••••

Sample Problem Statements:

1. Design and develop a UI for calculator using React.

You can include features like-

Mathematical operations

Inserting values

Decimal values must be supported

2. Design and develop a UI for creating React form to gather the individual's data.

It should include features like-

Taking user's data.

A SUBMIT button.

On submit must highlight the empty block.

Error message when entering the wrong format of email ID.

Display a success message on successful submission.

3. Design and develop a UI for Resume Builder application system.

The resume builder app must include -

Professional summary

Education qualifications

Academic and non-academic skills,

Career objective

Experience and Internships

Skills and Achievements