React JS

Rohan Chhetry

Components

- Components are the building blocks of a React application, representing UI elements.
- Components can be class-based or functional.

Components

Problem 1

Create a functional component **HelloWorld** that renders a simple greeting message.

Components

Solution 1

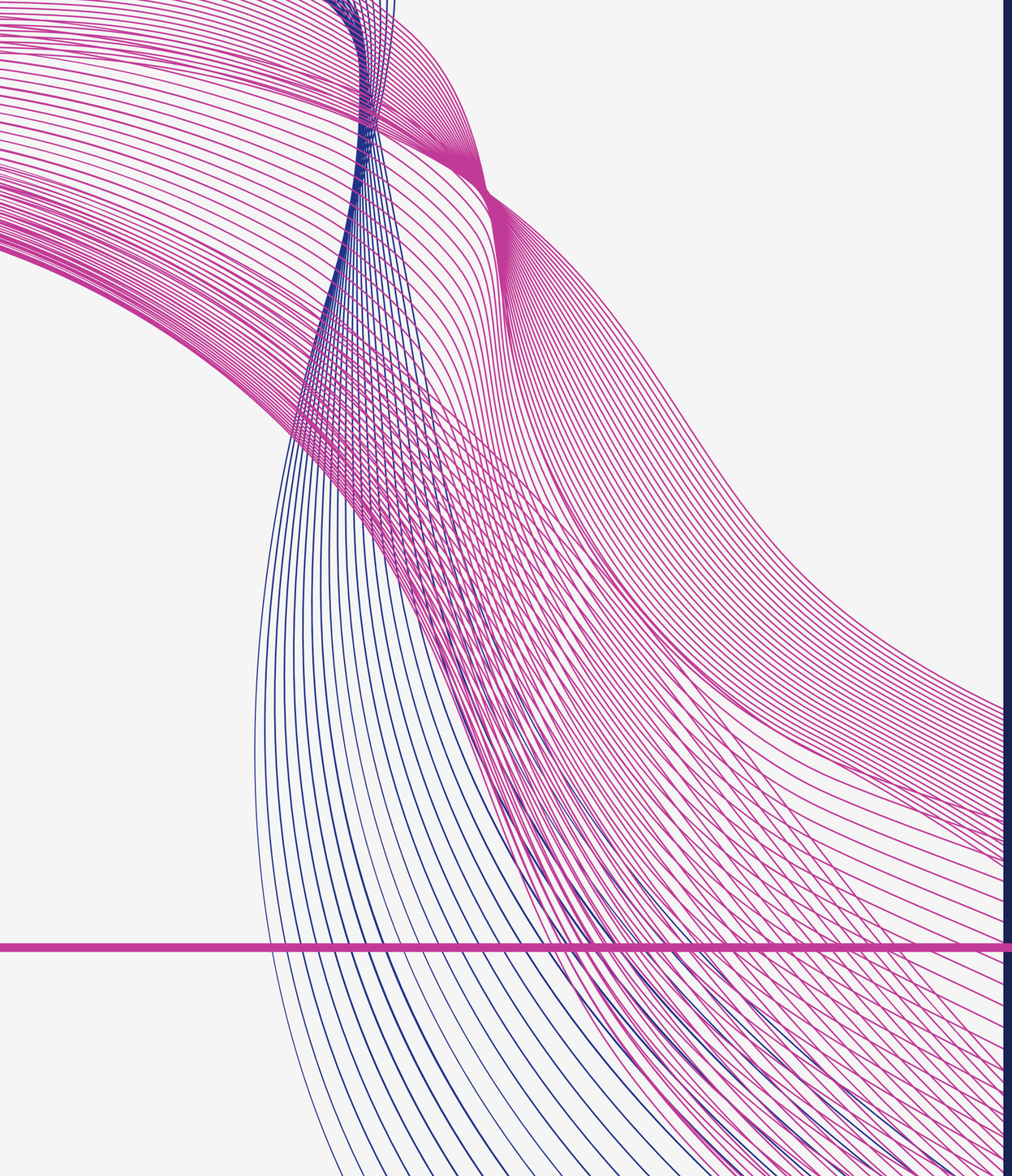
```
import React from 'react';

function HelloWorld() {
  return <div>Hello, World!</div>;
}

export default HelloWorld;
```

VIRTUAL DOM

- The Virtual DOM is a lightweight copy of the actual DOM, used to improve performance by minimizing direct manipulation of the real DOM.
- The Virtual DOM allows React to efficiently update only the changed parts of the actual DOM, reducing the need for expensive direct DOM manipulations.

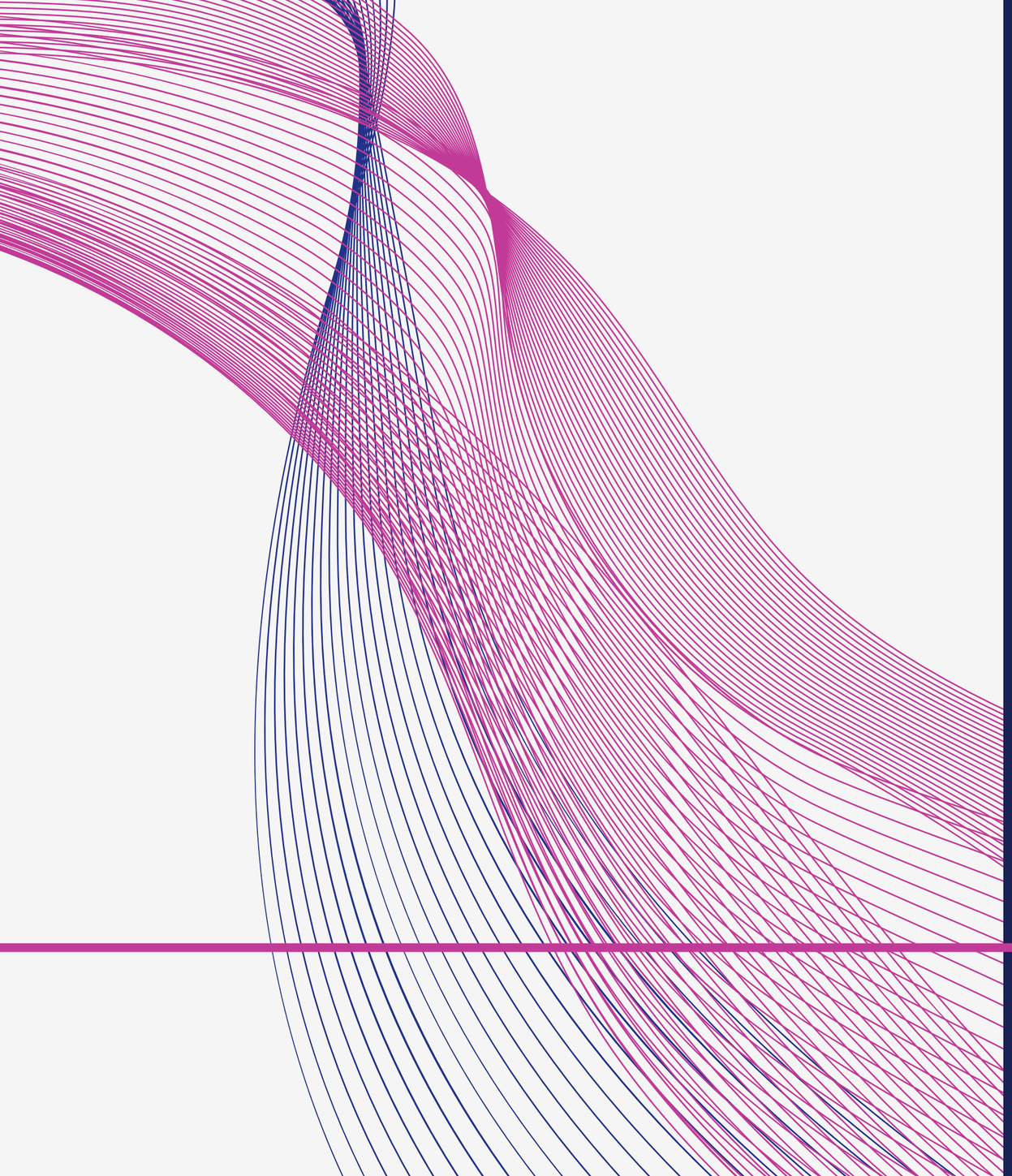


CLASS & FUNCTION COMPONENTS

- Class components are ES6 classes that extend React.Component.
- Functional components are simpler and recommended for simple UI elements.

PROBLEM 2

Convert the HelloWorld component to a class component.



CLASS & FUNCTION COMPONENTS

Solution 2

```
import React, { Component } from 'react';

class HelloWorldClass extends
Component {
  render() {
    return <div>Hello, World!</div>;
  }
}

export default HelloWorldClass;
```

JSX
JAVASCRIPT XML

JSX (JavaScript XML) is a syntax extension for JavaScript recommended for use with React to describe what the UI should look like.

Problem 3

Write JSX code to create a button element with the label "Click Me."

JSX
JAVASCRIPT XML

Solution 3

```
import React, { Component } from 'react';

class HelloWorldClass extends
Component {
  render() {
      const buttonLabel = "Click Me";
      const buttonElement = <button>{buttonLabel}</button>;
    return (
      <div>{buttonElement}</div>
    );
  }
}
```

Props

- Props (short for properties) are inputs passed into React components.
- Props allow components to be customizable and reusable.

Props Problem 4

Create a Greeting component that takes a name prop and displays a personalized greeting.

Props Solution 4

<Greeting name=”my_name”/>

```
import React from 'react';

function Greeting(props) {
  return <div>Hello, {props.name}!</div>;
}

export default Greeting;
```

Conditional Rendering

Conditional rendering allows components to display different content based on conditions.

```
const logIn = true;
<LoginStatus isLoggedIn={logIn}/>
```

Problem 5

Create a LoginStatus component that displays "Welcome" if the user is logged in and "Please log in" if not.

Solution 5

```
import React from 'react';

function LoginStatus({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <p>Welcome!</p> : <p>Please log in</p>}
    </div>
  );
}

export default LoginStatus;
```

State Management

01 State allows a component to maintain and manage its own data

02 State changes trigger re-rendering of the component.

Problem 6

Create a Counter component that increments a count when a button is clicked.

State Management

Solution 6

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

LIFECYCLE METHODS

- Lifecycle methods are methods that are invoked at various stages of a component's life.
- `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` are examples.

NPM

Problem 7

Explain how to initialize a new React project using npm.

Solution 7

npx create-react-app my-react-app
cd my-react-app
npm start

- **npm** (Node Package Manager) is a package manager for JavaScript, used for managing dependencies and scripts

Event Handling

Event handling in React involves using functions to respond to user interactions

Problem 8

Create a ClickButton component that logs a message when a button is clicked.

Solution 8

```
import React from 'react';

function ClickButton() {
  const handleClick = () => {
    console.log('Button clicked!');
  };

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

export default ClickButton;
```

FORM HANDLING

- FORM HANDLING INVOLVES MANAGING FORM INPUT AND SUBMISSION IN REACT

- **PROBLEM 9**
  CREATE A LOGINFORM COMPONENT THAT LOGS THE ENTERED USERNAME AND PASSWORD WHEN THE FORM IS SUBMITTED.

SOLUTION 9

```
IMPORT REACT, { USESTATE } FROM 'REACT';

FUNCTION LOGINFORM() {
  CONST [USERNAME, SETUSERNAME] = USESTATE('');
  CONST [PASSWORD, SETPASSWORD] = USESTATE('');

  CONST HANDLESUBMIT = (E) => {
    E.PREVENTDEFAULT();
    CONSOLE.LOG(`USERNAME: ${USERNAME}, PASSWORD: ${PASSWORD}`);
  };

  RETURN (
    <FORM ONSUBMIT={HANDLESUBMIT}>
                <INPUT TYPE="TEXT" PLACEHOLDER="USERNAME" VALUE={USERNAME} ONCHANGE={(E) =>
SETUSERNAME(E.TARGET.VALUE)} />
          <INPUT TYPE="PASSWORD" PLACEHOLDER="PASSWORD" VALUE={PASSWORD} ONCHANGE={(E) =>
SETPASSWORD(E.TARGET.VALUE)} />
      <BUTTON TYPE="SUBMIT">SUBMIT</BUTTON>
    </FORM>
  );
}

EXPORT DEFAULT LOGINFORM;
```

Debug the code , it will not work with all caps

HOOKS

Problem 10

Modify the Counter component to use the useEffect hook to log a message after each render.

- Hooks are functions that enable functional components to use state and other React features.
- Hooks include useState, useEffect, and others, providing additional capabilities to functional components

Solution 10

HOOKS

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

function Counter() {
  const [count, setCount] = useState(0);

useEffect(() => {
    console.log(`Component rendered. Count is ${count}`);
  }, [count]);


  const increment = () => setCount(count + 1);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
export default Counter;
```

REACT ROUTER

React Router is a library for navigation in React applications, enabling the creation of single-page applications with multiple views.

Problem 11

Set up a basic navigation system using React Router with two routes: /home and /about.

SOLUTION 11

```
// // npm install react-router-dom
import { BrowserRouter as Router, Route, Link, Routes } from 'react-router-dom';
// Home Component
function Home() {
  return <>
    <h2>Home</h2>
    <Link to="/about">About</Link><br />
    <Link to="/contact">Contact</Link>
  </>;
}
// About Component
function About() {
  return <>
    <h2>About</h2>
    <Link to="/">Home</Link><br />
    <Link to="/contact">Contact</Link>
  </>;
}
```

SOLUTION 11

```
// Contact Component
function Contact() {
 return <>
 <h2>Contact</h2>
 <Link to="/">Home</Link><br />
 <Link to="/about">About</Link>
 </>;
}

function App() {
 return (
 <Router>
 <Routes>
 <Route path="/" element={<Home />} />
 <Route path="/about" element={<About />} />
 <Route path="/contact" element={<Contact />} />
 </Routes>
 </Router>
 );
}

export default App;
```

Server-Side RENDERING

- Server-Side Rendering (SSR) involves rendering React components on the server rather than the client, improving initial load times.
- SSR improves the application's performance by delivering a fully rendered HTML page to the client, reducing initial load times and enhancing SEO.

REDUX

Redux is a state management library for React applications, providing a centralized state container

Main three components:

1. Action
2. Reducer
3. Store

REDUX

Actions are plain JavaScript objects that represent payloads of information that send data from your application to your Redux store. They are the only source of information for the store. Actions must have a type property that indicates the type of action being performed. They may also contain additional data.

Example:

```
{
 type: 'ADD_TODO',
 text: 'Learn Redux'
}
```

REDUX

Reducers specify how the application's state changes in response to actions sent to the store. Actions describe the fact that something happened, but they don't specify how the application's state changes in response. This is the job of reducers.

Reducers are pure functions that take the current state and an action, and return a new state. They should not mutate the original state but return a new object if the state needs to change.

REDUX

Example of reducer:

```
function todos(state = [], action) {
 switch (action.type) {
    case 'ADD_TODO':
      return [...state, { text: action.text,
completed: false }];
    default:
      return state;
 }
}
```

REDUX

The Redux **store** brings together the state, actions, and reducers. It holds the whole state tree of your application. The store has the following responsibilities:

- Holds application state;
- Allows access to state via getState();
- Allows state to be updated via dispatch(action);
- Registers listeners via subscribe(listener);
- Handles unregistering of listeners via the function returned by subscribe(listener).

You create a store by passing the reducer to the createStore function from Redux.

REDUX

Example:

```
import { createStore } from 'redux';
import todoApp from './reducers';
let store = createStore(todoApp);
```

REDUX

Problem 12

Implement a basic Redux setup with actions, a reducer, and a store for managing a counter.

SOLUTION 12

```
import React from 'react';
import { createStore } from 'redux';
import { Provider, connect } from 'react-redux';
// Actions

const increment = () => ({ type: 'INCREMENT' });
const decrement = () => ({ type: 'DECREMENT' });
// Reducer
const counterReducer = (state = 0, action) => {
 switch (action.type) {
  case 'INCREMENT':
   return state + 1;
  case 'DECREMENT':
   return state - 1;
  default:
   return state;
 }
};
```

```
// Store
const store = createStore(counterReducer);

// React Component
class Counter extends React.Component {
 render() {
 return (
 <div>
 <p>Count: {this.props.count}</p>
 <button onClick=
{this.props.increment}>Increment</button>
 <button onClick=
{this.props.decrement}>Decrement</button>
 </div>
 );
 }
}
```

SOLUTION 12

```
// Connect React component to Redux
const mapStateToProps = state => ({
 count: state
});

const mapDispatchToProps = dispatch => ({
 increment: () => dispatch(increment()),
 decrement: () => dispatch(decrement())
});

const ConnectedCounter =
connect(mapStateToProps,
mapDispatchToProps)(Counter);
```

```
// Render the app
const Rdx = () => (
 <Provider store={store}>
 <ConnectedCounter />
 </Provider>
);

export default Rdx;
```

Context API

Context API provides a way to share values (like themes or authentication status) across the component tree without passing props manually.

Problem 13

Create a ThemeContext to share the current theme across components.

Solution 13

```
import React, { createContext, useContext, useState } from 'react';

const ThemeContext = createContext();

function ThemedComponent() {
  const theme = useContext(ThemeContext);
  return <div style={{ color: theme }}>Themed Component</div>;
}

function Cntx() {
  const [theme, setTheme] = useState('blue');
  return (
    <ThemeContext.Provider value={theme}>
      <ThemedComponent />
      <button onClick={() => setTheme('red')}>Change Theme to Red</button>
      <button onClick={() => setTheme('blue')}>Change Theme to Blue</button>
    </ThemeContext.Provider>
  );
}
export default Cntx;
```

NEXT JS

Next.js is a framework for building server-rendered applications, providing features like routing and server-side rendering.

Problem 14

Create a simple Next.js application with a home page and an about page.

SOLUTION 14

Npx create-next-app my-app

```
// home/page.jsx
import Link from "next/link";
import React from 'react'

const Home = () => {
 return (
  <div>
   <h1>Home</h1>
   <Link href="/">Hello</Link><br />
   <Link href="/about">About</Link>
  </div>
 )
}

export default Home
```

```
// about/page.jsx
import Link from "next/link";
import React from 'react'

const About = () => {
 return (
  <>
   <h1>About</h1>
   <Link href="/">Hello</Link><br />
   <Link href="/home">Home</Link>
  </>
 )
}

export default About
```

Integration

with Other Libraries

React can be integrated with other libraries and tools, such as Redux for state management or Axios for making HTTP requests.

CONNECT WITH API

Connecting with an API involves making HTTP requests to fetch or send data

Solution 13

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function SimpleComponent() {
 const [data, setData] = useState(null);

 useEffect(() => {
  axios.get('https://jsonplaceholder.typicode.com/todos/1')
   .then((response) => {
    setData(response.data);
   })
   .catch((error) => {
    console.error(`Error fetching data: ${error}`);
   });
 }, []); // Empty array ensures effect runs once on mount

  return (
  <div>
  {data ? (
  <div>{JSON.stringify(data)}</div>
  ) : (
  <div>Loading...</div>
  )}
  </div>
  );
 }

 export default SimpleComponent;
```

Problem 13

Use Axios to fetch data from a public API and display it in a React component.

THE FLUX ARCHITECTURE

- The Flux architecture is a unidirectional data flow pattern commonly used with React to manage application state
- Flux architecture emphasizes unidirectional data flow, with actions triggering updates through a dispatcher to stores, ensuring predictable and manageable state changes.

PROVIDER API

The Provider component in React is part of the Context API and allows passing down values to components in the tree.

Dispatch Methods

Dispatch methods are used in Redux to trigger actions, updating the application state.

Problem 14

Implement a Redux action with a dispatch method to increment a counter.

SOLUTION 14

Refer to solution 12

Axions and its usage with REST API

Axios is a popular JavaScript library for making HTTP requests. It is often used in React applications for data fetching.

Problem 15

Use Axios to fetch data from a REST API in a React component

SOLUTION 15

Refer to solution 13