**Introduction to React.js**

1. **Objective:**

The objective of this report is to provide an overview and understanding of the basic concepts and components of React.js. React.js is a popular JavaScript library used for building user interfaces. In this report, we will explore the key concepts and features of React.js and demonstrate a basic React application.

1. **Summary:**

React.js is a declarative and component-based JavaScript library developed by Facebook. It allows developers to build interactive and dynamic user interfaces by breaking them down into reusable components. React follows a virtual DOM (Document Object Model) approach, which efficiently updates and renders the components based on the application state changes.

1. **Run a React.js Application**

To run a React application, you need to follow these steps:

1. Install Node.js: Ensure that Node.js is installed on your computer. You can download and install it from the official Node.js website (https://nodejs.org).
2. Create a React application: Open your terminal or command prompt and navigate to the desired directory where you want to create your React application. Use the following command to create a new React application:

**npx create-react-app my-app**

1. Navigate to the application directory: After the application is created, navigate to the application directory using the following command:

**cd my-app**

1. Start the development server: Once you are inside the application directory, run the following command to start the development server:

**npm start**

This command will start the development server and automatically open your application in a web browser. Any changes you make to the code will trigger a hot reload, meaning the application will automatically update in the browser without the need for a manual refresh.

1. Explore and modify the application: With the development server running, you can now start exploring and modifying your React application. Open your preferred code editor and navigate to the application's source code directory, which is located at src within your application directory. You can make changes to the existing files or create new components to build your application.
2. Build and deploy the application: Once you are satisfied with your application, you can build it for production using the following command:

**npm run build**

This command will create an optimized and minified version of your application in the build directory. You can then deploy the contents of the build directory to a web server or hosting platform to make your React application available online.

To stop the development server by pressing Ctrl + C in the terminal when you're finished working on your application.

1. **Key Concepts:**

**Components:** React applications are built using components, which are self-contained and reusable pieces of code that represent a part of the user interface. Components can be either functional or class-based.

**JSX (JavaScript XML):** JSX is an extension of JavaScript syntax that allows developers to write HTML-like code within JavaScript. It enables the creation of React elements in a more intuitive manner.

**State:** State represents the data that can change within a component. It helps in creating dynamic and interactive user interfaces by updating the state and triggering re-rendering of the components.

**Props (Properties):** Props are used to pass data from a parent component to its child components. They allow the components to be customizable and flexible.

1. **Basic React Application:**

As a practical example, a basic React application was created during the course of this study. The application consists of a single component that takes user input and displays a greeting message.

1. **Conclusion:**

In conclusion, this report provided an introduction to the basic concepts of React.js, including components, JSX, state, and props. It also demonstrated the creation of a basic React application. React.js offers a powerful and efficient way to build interactive user interfaces, making it a popular choice among developers. Understanding the fundamentals of React.js is essential for further exploration and development of more complex React applications.

**Creating Login and Registration Pages with React.js and Validation using Regex**

1. **Introduction:**

In this report, I will discuss the implementation of a login and registration page using React.js. Additionally, I will explore the use of regular expressions (regex) for form field validation to ensure the accuracy and security of user inputs.

1. **Login and Registration Pages:**

Using React.js, I developed two separate components for the login and registration pages. The login page consisted of email and password input fields, while the registration page included fields such as name, username, email, password, re-enter password, father's name, and mother's name. These components were structured using JSX syntax, allowing for the dynamic rendering of elements based on user interactions.

1. **Form Validation with Regex:**

To validate user inputs, I employed regular expressions, a powerful tool for pattern matching and data validation. For email validation, I used a regex pattern to ensure that the entered email followed the standard email format. Similarly, for password validation, I implemented a regex pattern that enforced password complexity rules, such as minimum length, uppercase and lowercase letters, digits, and special characters.

**Example**

const emailRegex = /^[^\s@][+@[^\s@]+\.[^\s@]+$/](mailto:+@[%5e\s@%5d+\.%5b%5e\s@%5d+$/);

let validationErrors = {};

// Validate email

if (!emailRegex.test(email)) {

validationErrors.email = "Please enter a valid email address";

}

<div>

<label>Email:</label>

<input

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

{errors.email && <span className="error">{errors.email}</span>}

</div>

1. **Client-Side Validation:**

Using React's useState hook, I created state variables to store the form data and error messages. As the user interacted with the form fields, I utilized event handlers to update the corresponding state variables and perform real-time validation using regex patterns. If the user input did not match the specified pattern, an error message was displayed to provide feedback and guide the user towards correct input.

1. **Form Submission and Error Handling:**

Upon form submission, I implemented a submit handler function that performed a final validation check before sending the data to the server. If any errors were detected during this process, the form submission was prevented, and the specific error messages were displayed to the user for correction. This ensured that only valid and properly formatted data was submitted to the backend.

1. **Styling and User Experience:**

To enhance the visual appearance and user experience, I utilized CSS and Bootstrap. I applied CSS styles to customize the layout, typography, colors, and spacing of the login and registration pages. By leveraging Bootstrap classes, I achieved responsive design, making the pages adapt to different screen sizes and devices.

1. **Conclusion:**

In conclusion, I successfully created login and registration pages using React.js, providing users with a seamless experience for authenticating and registering into the system. Through the implementation of form validation using regex, I enforced data integrity and security, ensuring that only valid and properly formatted information was accepted. The utilization of client-side validation, along with error handling and feedback mechanisms, improved the user experience by providing immediate feedback and guidance. By incorporating CSS and Bootstrap, I enhanced the visual appeal and responsiveness of the pages, resulting in an overall polished and professional design.