# MERN Stack Training

## Weekly Tasks

### Week 5:

#### 1. How to write your first React component

**Tasks**:

1. Create a simple functional component named Greeting that returns "Hello, World!" within a <h1> tag.
2. Modify the Greeting component to display "Hello, React!".
3. Create a Gallery functional component to display an image.
4. Add Greeting to the Gallery component and display the image and greeting.
5. Write a component called Profile which displays a hardcoded user's name and age.

---

#### 2. When and how to create multi-component files

**Tasks**:

1. Create a file named UserComponents.js and inside it, define two components: UserName and UserAge that display hardcoded names and ages respectively.
2. Export both UserName and UserAge from UserComponents.js.
3. In a separate file, import and use both UserName and UserAge components using named imports.
4. Convert UserAge into a default export and modify the importing file to accommodate the change.
5. Split UserName and UserAge into separate files and adjust your imports.

## 3. How to add markup to JavaScript with JSX

**Tasks**:

1. Create a component that displays an unordered list (<ul>) of 3 favorite fruits.
2. Update the above component to display a picture (<img>) of each fruit next to its name. (Use hardcoded image URLs for now.)
3. Create a component WebsiteLink that displays a hardcoded URL in an anchor (<a>) tag.
4. Make a JSX component that mimics a simple blog post with a title, content, and author. (All hardcoded.)
5. Design a Footer component with hardcoded copyright information using JSX.

## 4. JavaScript in JSX with Curly Braces

**Tasks**:

1. Display today's date in a component using the JavaScript Date object.
2. Create a component that displays a random quote from a hardcoded list of quotes.
3. Write a component called MathResult that displays the result of a simple arithmetic operation (e.g., addition) of two hardcoded numbers.
4. Create a component that displays the word count of a hardcoded paragraph.
5. Create a component that calculates and displays the product of two hardcoded numbers.

## 5. Passing Props to a Component

**Tasks**:

1. Create a Movie component that displays the title, year, and rating of a movie using props.
2. Update the Movie component to have a default prop for rating as “Not Rated”.
3. Design a Button component that takes in a label prop and displays the label on the button.
4. Make a UserProfile component and pass an object containing user details as props and display them.
5. Develop a Modal component that accepts and displays a title and some content passed as props.

---

## 6. Conditional Rendering

**Tasks**:

1. Design a UserStatus component that displays “Online” or “Offline” based on a isOnline prop.
2. Write a component AgeCheck that displays “Adult” or “Minor” based on an age prop.
3. Create a Loading component that either displays “Loading…” or content based on a isLoading prop.
4. Make a Notification component that conditionally displays a message if a message prop is provided.
5. Design a Feedback component that displays feedback in either green (positive) or red (negative) based on a type prop.

---

## 7. Rendering Lists

**Tasks**:

1. Write a component that takes an array of names as a prop and displays them in a list.
2. Create a `TodoList` component that displays a list of tasks and marks the completed ones.
3. Design a `ProductList` component that only displays products with a price less than $10 using the `filter()` method.
4. Make a `UserList` component that takes an array of user objects and displays their names and emails.
5. Create a `ShoppingCart` component that displays a list of items and their prices. Ensure each item has a unique key.

---

## 8. Keeping Components Pure

**Tasks**:

1. Convert an impure component that uses `Math.random()` within the render phase to a pure one.
2. Create a pure component `Clock` that displays the current time and updates every second without causing side-effects during the render phase.
3. Use Strict Mode in an existing application and identify any warnings in the console.
4. Convert a class-based component with side effects in its lifecycle methods to a pure functional component using hooks.
5. Make a pure `ProfilePic` component that takes a user ID as a prop and fetches the user's profile picture URL from an array without side-effects during rendering.

**Mini Project: "React Blog Portal"**

---

**Objective**: Build a simple blog portal where users can see a list of blog posts, view details of each post, and perform some basic filtering and interaction without the need for backend data.

---

## 1. Blog Post Component

- Create a BlogPost component to represent individual blog posts.
- Each post should display a title, content, and author.
- Use JSX to structure your post aesthetically.

---

## 2. Multi-component Blog

- Split the BlogPost into separate components:
    - PostTitle
    - PostContent
    - PostAuthor
- Store these components in a multi-component file and use named exports.

---

## 3. Dynamic Blog Content with JSX and JavaScript

- Use JavaScript arrays to store multiple blog post contents.

- Use JSX to display this dynamic content. For instance, format the date of the post using JavaScript's Date object.
- Display a random quote at the beginning of each post content from a list of predefined quotes.

---

### 4. Blog Listing and Details

- Create a main BlogList component that lists all the blog titles.
- When a user clicks on a blog title, display the full content of the blog post using conditional rendering.

---

### 5. Filtering Blog Posts

- Add a search bar to filter blog posts based on their titles.
- Use JavaScript's string methods to perform this filtering within the BlogList component.

---

### 6. Additional Features using Advanced Topics

- **Conditional Rendering**: Add a feature to hide or show the blog post content when the title is clicked.
- **Rendering Lists**: Display a list of related blog titles at the end of each blog post. Use JavaScript's map() to achieve this.

- **Keeping Components Pure**: Ensure that all your components remain pure. Any data manipulation should happen outside the component rendering logic.

---

**Project Workflow**:

1. Start by setting up a new React application using `create-react-app`.
2. Create the necessary components and their respective JSX structures.
3. Add the required logic for features like dynamic content rendering, conditional rendering, and list rendering.
4. Continuously test your application after each step to ensure everything works as expected.
5. Polish the user interface with some basic styling.

**Bonus**: Add the ability for users to add comments on a blog post, mimicking the interaction but without storing the comments permanently. Use local component state to temporarily store and display comments for each blog post.