Day 1: React Fundamentals (Basics) - 8 Hour Study Plan

📅 Daily Schedule

- **Teaching Hours:** 5 hours (Theory + Live Coding)
- **Practice Hours:** 3 hours (Assignments + Projects)
- **Total Duration:** 8 hours

🎯 Learning Objectives

By the end of Day 1, students will be able to:

- Understand React's core philosophy and virtual DOM
- Set up React development environment
- Write JSX syntax confidently
- Create functional and class components
- Pass data using props
- Manage component state with useState hook
- Handle user events in React applications

📚 Theory Section (3 Hours)

1. Introduction to React (45 minutes)

What is React?

React is a **declarative, efficient, and flexible JavaScript library** for building user interfaces, particularly web applications. Developed by Facebook (now Meta) in 2013.

Key Concepts:

- **Component-Based Architecture:** Build encapsulated components that manage their own state
- **Virtual DOM:** React creates an in-memory virtual representation of the real DOM
- **One-Way Data Flow:** Data flows down from parent to child components
- **Declarative Programming:** Describe what the UI should look like, not how to achieve it

Why React?

1. **Reusability:** Components can be reused across different parts of the application
2. **Performance:** Virtual DOM enables efficient updates

3. **Large Ecosystem:** Extensive library support and community
4. **Industry Adoption:** Used by Netflix, Airbnb, Instagram, WhatsApp
5. **Developer Experience:** Great tooling and debugging capabilities

Real-World Scenarios:

- **Facebook:** News feed, comments, reactions
- **Netflix:** Content browsing, user profiles
- **Airbnb:** Property listings, booking interface
- **WhatsApp Web:** Message interface, contact management

2. Environment Setup (30 minutes)

Method 1: Create React App (CRA)

```bash
# Install globally
npm install -g create-react-app

# Create new project
npx create-react-app my-react-app
cd my-react-app
npm start
```

Method 2: Vite (Recommended - Faster)

```bash
# Create with Vite
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

Method 3: Next.js (Full-Stack Framework)

```bash
npx create-next-app@latest my-next-app
cd my-next-app
npm run dev
```

Project Structure Understanding:

```
my-react-app/
├── public/
│   ├── index.html
│   └── favicon.ico
├── src/
│   ├── components/
│   ├── App.js
│   ├── index.js
│   └── index.css
├── package.json
└── README.md
```

3. JSX and Rendering Elements (45 minutes)

What is JSX?

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code in JavaScript files.

JSX Rules:

1. **Single Parent Element:** Must return one parent element
2. **Self-Closing Tags:** `<img />`, `<input />`, `<br />`
3. **className instead of class:** `className="my-class"`
4. **camelCase for attributes:** `onClick`, `onChange`
5. **Expressions in curly braces:** `{variable}`, `{function()}`

JSX Examples:

```
jsx
```

```
// Basic JSX
const element = <h1>Hello, World!</h1>;

// JSX with expressions
const name = "John";
const greeting = <h1>Hello, {name}!</h1>;

// JSX with attributes
const image = <img src="logo.png" alt="Logo" className="logo" />;

// JSX with conditional rendering
const isLoggedIn = true;
const message = (
  <div>
    {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log in</h1>}
  </div>
);

// JSX with arrays/lists
const numbers = [1, 2, 3, 4, 5];
const listItems = (
  <ul>
    {numbers.map(number => <li key={number}>{number}</li>)}
  </ul>
);
```

Behind the Scenes:

```jsx
// JSX
const element = <h1 className="greeting">Hello, world!</h1>;

// Compiled JavaScript
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

4. Components Deep Dive (90 minutes)

Function Components (Modern Approach)

```jsx
```

```jsx
// Basic Function Component
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Arrow Function Component
const Welcome = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};

// Implicit Return
const Welcome = (props) => <h1>Hello, {props.name}!</h1>;
```

Class Components (Legacy but Important to Know)

jsx

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

class Welcome extends Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

Component Composition:

jsx

```jsx
function App() {
  return (
    <div>
      <Welcome name="Alice" />
      <Welcome name="Bob" />
      <Welcome name="Charlie" />
    </div>
  );
}
```

5. Props (Properties) - Data Communication (45 minutes)

What are Props?

Props are arguments passed into React components. They are read-only and help make components reusable.

Props Examples:

```jsx
// Parent Component
function App() {
  const user = {
    name: "John Doe",
    age: 25,
    email: "john@example.com"
  };

  return (
    <div>
      <UserCard
        name={user.name}
        age={user.age}
        email={user.email}
        isActive={true}
      />
    </div>
  );
}

// Child Component
function UserCard(props) {
  return (
    <div className="user-card">
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
      <p>Email: {props.email}</p>
      <p>Status: {props.isActive ? "Active" : "Inactive"}</p>
    </div>
  );
}
```

Destructuring Props:

```jsx
```

```
// Instead of props.name, props.age, etc.
function UserCard({ name, age, email, isActive }) {
  return (
    <div className="user-card">
      <h2>{name}</h2>
      <p>Age: {age}</p>
      <p>Email: {email}</p>
      <p>Status: {isActive ? "Active" : "Inactive"}</p>
    </div>
  );
}
```

Default Props:

```jsx
function Button({ text, color, size }) {
  return (
    <button
      className={`btn btn-${color} btn-${size}`}
    >
      {text}
    </button>
  );
}

// Default values
Button.defaultProps = {
  text: "Click Me",
  color: "primary",
  size: "medium"
};
```

6. State Management with useState (45 minutes)

What is State?

State is data that can change over time. When state changes, the component re-renders to reflect the new state.

useState Hook:

```jsx
```

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

function Counter() {
  // Declare state variable
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Multiple State Variables:

```
jsx
```

```
function UserForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [age, setAge] = useState(0);

  return (
    <form>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
      />
      <input
        type="number"
        value={age}
        onChange={(e) => setAge(parseInt(e.target.value))}
        placeholder="Age"
      />
    </form>
  );
}
```

Object State:

```
jsx
```

```
function UserProfile() {
  const [user, setUser] = useState({
    name: '',
    email: '',
    age: 0
  });

  const updateUser = (field, value) => {
    setUser(prevUser => ({
      ...prevUser,
      [field]: value
    }));
  };

  return (
    <div>
      <input
        value={user.name}
        onChange={(e) => updateUser('name', e.target.value)}
      />
      <input
        value={user.email}
        onChange={(e) => updateUser('email', e.target.value)}
      />
      <input
        value={user.age}
        onChange={(e) => updateUser('age', parseInt(e.target.value))}
      />
    </div>
  );
}
```

7. Event Handling (30 minutes)

Common Events:

```
jsx
```

```
function EventExamples() {
  const [message, setMessage] = useState('');

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

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('Form submitted!');
  };

  const handleChange = (e) => {
    setMessage(e.target.value);
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter') {
      console.log('Enter key pressed!');
    }
  };

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

      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={message}
          onChange={handleChange}
          onKeyPress={handleKeyPress}
        />
        <button type="submit">Submit</button>
      </form>
    </div>
  );
}
```

💻 Practical Section (2 Hours Live Coding)

Live Coding Session 1: Hello React App (30 minutes)

```jsx
```

```
// App.js
import React from 'react';
import './App.css';

function App() {
  const appName = "My First React App";
  const currentYear = new Date().getFullYear();

  return (
    <div className="App">
      <header className="App-header">
        <h1>{appName}</h1>
        <p>Welcome to React development!</p>
        <p>Copyright © {currentYear}</p>
      </header>
    </div>
  );
}

export default App;
```

Live Coding Session 2: Counter App (45 minutes)

```
jsx
```

```
// Counter.js
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  const [step, setStep] = useState(1);

  const increment = () => setCount(count + step);
  const decrement = () => setCount(count - step);
  const reset = () => setCount(0);

  return (
    <div className="counter">
      <h2>Counter App</h2>
      <div className="counter-display">
        <h1>{count}</h1>
      </div>

      <div className="counter-controls">
        <button onClick={decrement}>-</button>
        <button onClick={increment}>+</button>
        <button onClick={reset}>Reset</button>
      </div>

      <div className="step-control">
        <label>
          Step:
          <input
            type="number"
            value={step}
            onChange={(e) => setStep(parseInt(e.target.value) || 1)}
          />
        </label>
      </div>
    </div>
  );
}

export default Counter;
```

Live Coding Session 3: Digital Clock (45 minutes)

```
jsx
```

```
// DigitalClock.js
import React, { useState, useEffect } from 'react';

function DigitalClock() {
  const [time, setTime] = useState(new Date());

  useEffect(() => {
    const timer = setInterval(() => {
      setTime(new Date());
    }, 1000);

    return () => clearInterval(timer); // Cleanup
  }, []);

  const formatTime = (date) => {
    return date.toLocaleTimeString('en-US', {
      hour12: true,
      hour: '2-digit',
      minute: '2-digit',
      second: '2-digit'
    });
  };

  const formatDate = (date) => {
    return date.toLocaleDateString('en-US', {
      weekday: 'long',
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    });
  };

  return (
    <div className="digital-clock">
      <h2>Digital Clock</h2>
      <div className="clock-display">
        <div className="time">{formatTime(time)}</div>
        <div className="date">{formatDate(time)}</div>
      </div>
    </div>
  );
}

export default DigitalClock;
```

🔧 Assignment Section (3 Hours Practice)

Assignment 1: Personal Info Card (45 minutes)

Objective: Create a reusable PersonCard component that displays personal information.

Requirements:

- Create a PersonCard component that accepts props for name, age, profession, and hobbies
- Display a profile picture (use a placeholder)
- Add a "Show/Hide Details" button that toggles additional information
- Style the card to look professional

```jsx
// Starter code structure
function PersonCard({ name, age, profession, hobbies, profileImage }) {
  // Your implementation here
}

function App() {
  const people = [
    {
      name: "Alice Johnson",
      age: 28,
      profession: "Software Engineer",
      hobbies: ["Reading", "Cooking", "Hiking"],
      profileImage: "https://via.placeholder.com/150"
    },
    // Add more people
  ];

  return (
    <div>
      {/* Render PersonCard components */}
    </div>
  );
}
```

Assignment 2: Interactive Todo Input (60 minutes)

Objective: Build a todo input component with validation and dynamic features.

Requirements:

- Input field for todo text

- Add button (disabled when input is empty)
- Character counter (max 100 characters)
- Clear button
- Input validation with error messages
- List display of added todos

```jsx
// Expected features
function TodoInput() {
  // State management for:
  // - input value
  // - todos array
  // - error messages
  // - character count

  // Functions to implement:
  // - handleInputChange
  // - addTodo
  // - clearInput
  // - validateInput
}
```

Assignment 3: Theme Switcher Component (45 minutes)

Objective: Create a theme switcher that changes the app's appearance.

Requirements:

- Button to toggle between light and dark themes
- Apply theme to multiple components
- Store theme preference in component state
- Smooth theme transitions

Assignment 4: Calculator Component (50 minutes)

Objective: Build a basic calculator with essential operations.

Requirements:

- Number buttons (0-9)
- Operation buttons (+, -, *, /)
- Clear and equals buttons
- Display for current number and result

- Handle basic error cases (division by zero)

🎯 Project: Advanced Digital Clock App

Project Requirements (Extended Version):

1. **Multiple Time Zones:** Display time for different cities
2. **Alarm Feature:** Set and trigger alarms
3. **Stopwatch:** Start, stop, reset functionality
4. **Timer:** Countdown timer with notifications
5. **Theme Customization:** Multiple color themes
6. **Settings:** Toggle between 12/24 hour format

Project Structure:

```jsx
// Components to create:
// - DigitalClock (main component)
// - TimeZoneSelector
// - AlarmSetter
// - Stopwatch
// - Timer
// - ThemeSelector
// - Settings
```

🧠 Logical Thinking Questions

Question 1: Component Re-rendering

Scenario: You have a parent component with multiple child components. When you update state in the parent, all children re-render even though only one child needs the updated data.

Challenge: How would you optimize this to prevent unnecessary re-renders?

Question 2: State Management

Scenario: You have a form with 10 input fields. Currently, you're using 10 separate useState hooks.

Challenge: Discuss the pros and cons of this approach vs. using a single state object. When would you choose one over the other?

Question 3: Event Handling

Scenario: You have a list of 1000 items, each with a delete button.

Challenge: How would you efficiently handle click events without creating 1000 separate event handlers?

📝 MCQ Questions (Self-Assessment)

Question 1:

What will be the output of this code?

```jsx
function App() {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
    setCount(count + 1);
    setCount(count + 1);
  };

  return <button onClick={handleClick}>{count}</button>;
}
```

A) Increases by 3 each click
B) Increases by 1 each click
C) Increases by 2 each click
D) Causes an error

Answer: B - State updates are batched and use the same `count` value

Question 2:

Which of these is NOT a valid way to pass props?

```jsx
A) <Component name="John" />
B) <Component name={userName} />
C) <Component {user} />
D) <Component {...userProps} />
```

Answer: C - Invalid syntax for prop spreading

Question 3:

What's the correct way to update an object in state?

```jsx
const [user, setUser] = useState({name: 'John', age: 25});

A) user.age = 26; setUser(user);
B) setUser({age: 26});
C) setUser({...user, age: 26});
D) setUser(user => user.age = 26);
```

Answer: C - Must spread existing properties and update specific ones

---

🔑 Key Points Summary

Essential Concepts to Remember:

1. **Virtual DOM:** React's efficiency comes from virtual DOM diffing
2. **Unidirectional Data Flow:** Data flows from parent to child via props
3. **State Immutability:** Never mutate state directly, always use setState
4. **Component Lifecycle:** Understand when components mount, update, and unmount
5. **Event Handling:** Use arrow functions or bind methods properly
6. **JSX Rules:** Single parent, self-closing tags, camelCase attributes
7. **Props are Read-Only:** Components must never modify their own props
8. **Key Prop:** Always provide unique keys when rendering lists

Best Practices:

- Use functional components over class components
- Destructure props for cleaner code
- Use meaningful component and variable names
- Keep components small and focused
- Separate concerns (logic, presentation, styling)
- Use proper folder structure for larger applications

Common Pitfalls:

- Mutating state directly
- Using array indices as keys

- Missing key props in lists
- Forgetting to bind event handlers (in class components)
- Not cleaning up side effects

📚 Additional Resources

GitHub Repositories to Clone:

1. **React Official Examples:** https://github.com/facebook/react/tree/master/packages/react-dom/src/__tests__/fixtures
2. **React Patterns:** https://github.com/chantastic/reactpatterns.com
3. **React Hooks Examples:** https://github.com/rehooks/awesome-react-hooks

Recommended Reading:

- React Official Documentation: https://react.dev
- JavaScript ES6+ Features for React
- Component Design Patterns
- State Management Best Practices

Tools to Install:

- React Developer Tools (Chrome/Firefox extension)
- VS Code with ES7+ React/Redux/React-Native snippets
- Prettier for code formatting
- ESLint for code quality

🏆 Daily Achievement Checklist

By the end of Day 1, students should be able to:

- [ ] Set up a React development environment
- [ ] Create and export React components
- [ ] Write proper JSX syntax
- [ ] Pass and use props effectively
- [ ] Manage component state with useState
- [ ] Handle user interactions with event handlers
- [ ] Build a working digital clock application
- [ ] Understand React's rendering behavior
- [ ] Debug React applications using developer tools

Next Day Preview: Tomorrow we'll dive into **Lists and Keys, Conditional Rendering, and Component Lifecycle** - building more dynamic and interactive applications!