Lab 1: React Fundamentals - Building Your First Components

Practical Exercises for Chapter 1 (VITE Edition)

Table of Contents

Lab Overview

🎯 Lab Objectives

By completing this lab, you will:

- ☑ Create functional React components
- ☑ Understand JSX syntax and rules
- ☑ Work with props to pass data between components
- ☑ Compose components to build a UI
- ☑ Apply CSS styling to components
- ☑ Debug common React errors

📋 Prerequisites

- Node.js installed
- VS Code or your preferred editor
- Basic JavaScript knowledge (variables, functions, arrays, objects)
- Completed reading Chapter 1

🛠 Setup (VITE - Modern & Fast)

```
# Create a new React app with Vite (NOT create-react-app!)
npm create vite@latest lab-1-fundamentals -- --template react

# Enter the project
cd lab-1-fundamentals

# Install dependencies
npm install
```

```
# Start the development server
npm run dev
```

Your browser will open at `http://localhost:5173` (Vite default port).

Important: Notice the project structure:

- `index.html` is at ROOT level (not in `public/`)
- `src/main.jsx` is the entry point (not `src/index.js`)
- Component files use `.jsx` extension (e.g., `Greeting.jsx`, not `Greeting.js`)

Part 1: JSX and Components Basics

🎓 Learning Recap

Before starting, remember:

- Components are JavaScript functions that return JSX
- JSX looks like HTML but is JavaScript
- JSX gets compiled to `React.createElement()` calls
- Components must return a single root element
- **Use `.jsx` extension for files that contain React components!**

Exercise 1.1: Your First Component

Task: Create a simple greeting component

Instructions:

1. Create a new file: `src/components/Greeting.jsx` (notice: `.jsx` not `.js`)
2. Write a component that displays:
    - A greeting message: "Welcome to React!"
    - A subtitle: "This is my first component"
    - A fun fact about React
3. This component should NOT use props yet (hardcoded text is fine)
4. Export the component
5. Import and use it in `src/App.jsx`

Requirements:

- ☑ Component must be a function
- ☑ Component must return JSX
- ☑ Use semantic HTML tags (h1, h2, p)
- ☑ Add some basic CSS styling (you can use inline styles or CSS file)

Hints:

- Look at Chapter 1, Section 7 for component syntax
- Remember: Component names must start with capital letters
- Use `export default Greeting` to export

Testing your work:

- Does the component render on the page?
- Can you see all three pieces of information?
- Is it styled nicely?

Exercise 1.2: JSX Rules

Task: Fix JSX errors

Challenge: You have 3 broken JSX code snippets. Fix them!

Instructions:

1. Create a file: `src/exercises/JSXErrors.jsx` (notice: `.jsx` extension!)
2. Below are 3 broken components. Don't just copy them—write corrected versions:

```
// ERROR 1: What's wrong here?
function BadComponent1() {
  return (
    <h1>Hello</h1>
    <p>This is broken</p>
  )
}

// ERROR 2: What's wrong here?
function BadComponent2() {
  const isTrue = true
  return (
    <div>
      <p>Result: {if (isTrue) { 'Yes' }}</p>
    </div>
  )
}

// ERROR 3: What's wrong here?
function BadComponent3() {
  return (
    <div class="container">
      <img src="cat.jpg">
      <p>A paragraph</p>
    </div>
  )
}
```

What you need to do:

- Identify the error in each component
- Write the corrected version
- Create a comment above each corrected component explaining the error

Hints:

- Error 1: Think about "single root element" (use `<>` and `</>` or a `<div>`)
- Error 2: Think about expressions vs statements (use ternary operator)
- Error 3: Think about `className` vs `class` and self-closing tags

Testing:

- Can you explain why each was broken?
- Do your fixed versions not throw errors?

Exercise 1.3: Conditional Rendering

Task: Create a component that shows/hides content

Instructions:

1. Create: `src/components/StatusBadge.jsx` (`.jsx` extension!)
2. Create a component that:
    - Accepts a boolean value: `isOnline`
    - If true: shows "🟢 Online" with green styling
    - If false: shows "🔴 Offline" with red styling
    - Displays a message: "User is currently [online/offline]"
3. Use this component in `App.jsx` with BOTH states (create two instances: one with `true`, one with `false`)

Don't write props yet! Use hardcoded boolean values:

```
// At the top of the component
const isOnline = true // You'll change this value to test both cases
```

Requirements:

- ☑ Conditional rendering (ternary operator)
- ☑ Different styles for each state
- ☑ Works with both true and false

Hints:

- Use ternary operator: `isOnline ? <content if true> : <content if false>`

- Or use logical AND: `isOnline && <content>`
- Style with inline styles or CSS classes
- Test both true and false states

Testing:

- Does it show the correct icon and message for true?
- Does it show the correct icon and message for false?
- Are the colors different?

Part 2: Working with Props

🎓 Learning Recap

Remember:

- Props are arguments passed to components
- Props are immutable (read-only)
- Props allow component reusability
- Destructuring props makes code cleaner

Exercise 2.1: Simple Props

Task: Create a reusable User Card component

Instructions:

1. Create: `src/components/UserCard.jsx` (`.jsx` extension!)
2. This component should accept these props:
    - `name` (string): User's full name
    - `email` (string): User's email address
    - `role` (string): User's role (e.g., "Developer", "Designer")
3. Display these in a nicely formatted card:

```
[Name]
Email: [Email]
Role: [Role]
```

4. Use the component in `App.jsx` to create 3 different user cards:
    - User 1: Alice, alice@example.com, Developer
    - User 2: Bob, bob@example.com, Designer
    - User 3: Charlie, charlie@example.com, Manager

Requirements:

- ☑ Use destructuring in the function parameter
- ☑ Component works with different props
- ☑ Nice styling (like a card with border/shadow)

Hints:

- Look at Chapter 1, Section 7 for props syntax
- Try destructuring: `function UserCard({ name, email, role })`
- Use CSS to make it look like a card (create `UserCard.css`)

Testing:

- Does each card show the correct information?
- Can you change the props and see different data?
- Does it look like a proper card?

Exercise 2.2: Props with Different Data Types

Task: Create a Product component that accepts different data types

Instructions:

1. Create: `src/components/Product.jsx` (`.jsx` extension!)
2. This component accepts:
    - `title` (string): Product name
    - `price` (number): Price in dollars
    - `inStock` (boolean): Is it available?
    - `rating` (number): Star rating (0-5)
3. Display:
    - Product title as heading
    - Price formatted as "$XX.XX"
    - Stock status: "In Stock" or "Out of Stock" with different colors
    - Star rating: Show as "★★★★★" based on the number
4. Create 3 products in `App.jsx`:
    - Laptop: $999, in stock, 4.5 stars
    - Phone: $499, out of stock, 4 stars
    - Headphones: $99, in stock, 5 stars

Requirements:

- ☑ Handle number type (format price)
- ☑ Handle boolean type (conditional styling)
- ☑ Create correct number of stars based on rating

Hints:

- For price: `price.toFixed(2)`
- For stars: Use a loop or `.map()` with an array
- For stock status: Use conditional rendering

Testing:

- Are prices formatted correctly?
- Are stock statuses color-coded?
- Do the stars match the rating?

Exercise 2.3: Props with Children

Task: Create a Card wrapper component

Instructions:

1. Create: `src/components/Card.jsx` (`.jsx` extension!)
2. This component:
    - Accepts a `title` prop
    - Accepts `children` (content between opening/closing tags)
    - Displays the title as a header
    - Displays the children inside a styled card
3. Use it in `App.jsx` to create 3 cards with different content:

```
<Card title="Card 1">
  <p>This is the content inside Card 1</p>
</Card>
```

Requirements:

- ☑ Use the `children` prop
- ☑ Title and content are flexible
- ☑ Card has a border/shadow styling

Hints:

- `children` is automatically a prop
- You can access it like any other prop: `props.children`
- Or destructure it: `function Card({ title, children })`

Testing:

- Does the title appear for each card?
- Does the content inside appear correctly?
- Can you put different content in each card?

Part 3: Component Composition

🎓 Learning Recap

Component composition means:

- Breaking UI into small, reusable pieces
- Combining components to build larger components
- Parent components pass props to child components

Exercise 3.1: Building a Blog Post Component

Task: Create a complete blog post using smaller components

Instructions:

1. You already have a `UserCard` component. Now create:
2. `src/components/BlogPost.jsx` (`.jsx` extension!) that combines:
    - A header with the post title
    - The `UserCard` component showing the author
    - The post content (just text)
    - A footer with the date
3. Example structure:

```
[Post Title - as h1]
[Author info using UserCard - showing only name]
[Post content paragraph]
[Published date]
```

4. In `App.jsx`, create 2 blog posts with:
    - Post 1: Title "Learning React", Author: Alice (alice@example.com, Developer), Date: Jan 15
    - Post 2: Title "React Tips", Author: Bob (bob@example.com, Designer), Date: Jan 20

Requirements:

- ☑ Reuse the UserCard component
- ☑ Pass props from BlogPost to UserCard
- ☑ Style it nicely (padding, margins, fonts)

Hints:

- Import UserCard: `import UserCard from './UserCard'`
- Pass props to UserCard: `<UserCard name={...} email={...} role={...} />`
- Create a separate CSS file: `BlogPost.css`

Testing:

- Does the post title appear?
- Does the UserCard show author info?
- Does the post content display?
- Is the date visible?

Exercise 3.2: Building a Movie List

Task: Create a movie list using component composition

Instructions:

1. Create `src/components/Movie.jsx` (`.jsx` extension!):
   - Accepts: `title`, `director`, `year`, `rating`
   - Displays: Movie title, director, year, rating (stars)
2. Create `src/components/MovieList.jsx` (`.jsx` extension!):
   - Accepts: `movies` (array of movie objects)
   - Renders multiple `Movie` components, one for each movie using `.map()`
3. In `App.jsx`:
   - Create an array of 4 movies
   - Pass it to `MovieList`

Example movie data structure:

```
const movies = [
  { id: 1, title: "The Matrix", director: "Wachowski", year: 1999, rating: 4 },
  { id: 2, title: "Inception", director: "Nolan", year: 2010, rating: 5 },
  { id: 3, title: "The Dark Knight", director: "Nolan", year: 2008, rating: 5 },
  { id: 4, title: "Interstellar", director: "Nolan", year: 2014, rating: 5 }
]
```

Requirements:

- ☑ Movie component displays individual movie
- ☑ MovieList component uses `.map()` to render multiple movies
- ☑ Each movie has consistent styling
- ☑ Each movie item has a `key` prop (use the id)

Hints:

- Use `.map()` in MovieList
- Key prop: `<Movie key={movie.id} {...props} />`
- Movie component should show all 4 pieces of info

Testing:

- Do all movies appear in the list?
- Does each movie show correct info?
- Are they styled consistently?

Exercise 3.3: State Preview (Preparing for Chapter 2)

Task: Create a toggle component without state (just prepare the UI)

Instructions:

1. Create `src/components/ToggleButton.jsx` (`.jsx` extension!):
    - Shows a button with text "Click me!"
    - Has a content area below that shows:
        - "Content is VISIBLE" (with green styling)
        - OR "Content is HIDDEN" (with gray styling)
2. For now: Just create the UI. The button won't work yet (no state).
    - Create two versions using conditional rendering:
    - In `App.jsx`, show both the "visible" and "hidden" versions
3. Next chapter you'll make it actually toggle!

Requirements:

- ☑ Button element (doesn't need to work yet)
- ☑ Two different displays based on a variable
- ☑ Nice styling for visible/hidden states

Hints:

- Use a hardcoded boolean variable
- Test by changing the variable and restarting the dev server
- This prepares you for `useState` in Chapter 2

Testing:

- Does it show the visible message?
- Can you change the variable to show hidden?
- Is the styling different for each state?

Part 4: Project Challenge

🏆 Challenge: Personal Portfolio Preview

Difficulty: Medium

Time Estimate: 2-3 hours

Challenge Description

Build a **Personal Portfolio Website Preview** component. This is a single-page preview of what a portfolio might look like. Later chapters will make it interactive and add real functionality!

Requirements

Must Include (Core Features):

1. **Header Component** (`Header.jsx`)
   - Your name (as title)
   - A brief tagline about yourself
   - Navigation links (just display them, no routing yet):
     - Home
     - About
     - Projects
     - Contact

2. **About Section Component** (`About.jsx`)
   - A profile image (use a placeholder image URL)
   - A brief bio (2-3 sentences about yourself)
   - Your skills (just a list: React, JavaScript, HTML, CSS, etc.)

3. **Project Showcase Component** (`ProjectCard.jsx` and `ProjectShowcase.jsx`)
   - Display 3 projects as cards
   - Each project card shows:
     - Project name
     - Description
     - Technologies used (as a list)
     - An "amazing" emoji indicator (⭐)

4. **Contact Section Component** (`Contact.jsx`)
   - Your email
   - GitHub link (just the URL, no actual link yet)
   - LinkedIn link (just the URL)

5. **Footer Component** (`Footer.jsx`)
   - Copyright notice: "© 2024 [Your Name]"

File Structure You Should Have

```
src/
├── components/
│   ├── Header/
```

```
│   │   ├── Header.jsx          ☑ .jsx extension
│   │   └── Header.css
│   ├── About/
│   │   ├── About.jsx           ☑ .jsx extension
│   │   └── About.css
│   ├── ProjectCard/
│   │   ├── ProjectCard.jsx     ☑ .jsx extension
│   │   └── ProjectCard.css
│   ├── ProjectShowcase/
│   │   ├── ProjectShowcase.jsx ☑ .jsx extension
│   │   └── ProjectShowcase.css
│   ├── Contact/
│   │   ├── Contact.jsx         ☑ .jsx extension
│   │   └── Contact.css
│   ├── Footer/
│   │   ├── Footer.jsx          ☑ .jsx extension
│   │   └── Footer.css
├── App.jsx                     ☑ .jsx extension
├── App.css
├── main.jsx                    ☑ Entry point (already exists)
└── index.css
```

Step-by-Step Guide (Don't read ahead!)

Step 1: Plan your components

- Don't code yet! On paper or in comments, sketch:
  - What props does each component need?
  - What data is hardcoded vs. passed as props?
  - How do components relate to each other?

Step 2: Create Header component (`src/components/Header/Header.jsx`)

- Should accept: `name` (your name), `tagline` (your tagline)
- Display: Name as h1, tagline as h2
- Style it nicely with background color

Step 3: Create About component (`src/components/About/About.jsx`)

- Should accept: `profileImage` (URL), `bio` (text), `skills` (array)
- Display: Image, bio, skills as a list
- Hint: Use `.map()` for skills array

Step 4: Create Project components

- Create `ProjectCard.jsx` that accepts: `name`, `description`, `technologies`
- Create `ProjectShowcase.jsx` that:
  - Has an array of projects
  - Uses `.map()` to render ProjectCard for each
  - Hint: Don't hardcode projects in Showcase, receive them as props

Step 5: Create Contact component (`src/components/Contact/Contact.jsx`)

- Accept: `email`, `github`, `linkedin`
- Display: All contact info nicely formatted

Step 6: Create Footer component (`src/components/Footer/Footer.jsx`)

- Accept: `name`
- Display: Copyright notice

Step 7: Compose in App.jsx

- Import all components
- Create an `App` component that uses all of them
- Arrange them in a logical order

Step 8: Style everything

- Use CSS files or inline styles
- Make it look professional
- Use consistent colors, fonts, spacing

Challenge Questions (Answer these!)

1. **Component Reusability:**
    - Which components could be reused elsewhere? Why?
    - How did props help make components flexible?
2. **Data Flow:**
    - How does data flow from App.jsx to the smallest components?
    - Why not hardcode all data in each component?
3. **Composition:**
    - How did breaking the portfolio into components help?
    - What would happen if you tried to build it all in one component?

Stretch Goals (Optional - If you finish early)

- [ ] Add an avatar image that shows a circle (use CSS `border-radius: 50%`)
- [ ] Add a skills section with skill categories (e.g., "Frontend", "Tools")
- [ ] Add a "featured project" that stands out differently than others
- [ ] Add testimonials section (show 2 fake testimonials with quotes)
- [ ] Make the navigation links styled like buttons
- [ ] Add a background image or gradient to the header
- [ ] Add a theme toggle button (doesn't need to work yet)

Grading Rubric

Category	Excellent	Good	Fair	Needs Work
Component Structure	6+ components, well-organized	5 components, organized	4 components, somewhat organized	<4 components or disorganized
Props Usage	All components accept appropriate props	Most components use props	Some components hardcode data	All data hardcoded
JSX Quality	Semantic HTML, no errors, `.jsx` files	Valid JSX, few issues	Some JSX errors	Many JSX errors
File Extensions	All React files use `.jsx`	Most files use `.jsx`	Some files use `.jsx`	Files use wrong extensions
Composition	Components reuse other components	Some component nesting	Minimal reuse	No composition
Styling	Professional, consistent, responsive	Nice styling, mostly consistent	Basic styling, inconsistent	Minimal styling
Functionality	All requirements met, stretches included	All requirements met	Most requirements met	Missing requirements

Submission Guidelines

What to Submit

1. **Your code** (entire project folder)
2. **A README.md file** with:
    - **Your Name** and **Date**
    - **Lab Summary** (2-3 sentences about what you built)
    - **Components Created** (list all components and what they do)
    - **Setup Instructions:**

```
npm create vite@latest lab-1-fundamentals -- --template react
cd lab-1-fundamentals
npm install
npm run dev
```

    - **Challenges Encountered** (2-3 things that were tricky and how you solved them)
    - **Key Learnings** (3-4 things you learned)
    - **Vite Notes** (if any issues with Vite vs CRA)

3. **A Screenshot** of your working portfolio in the browser

Submission Format

```
Lab-1-Submission/
├── lab-1-fundamentals/          # Your Vite React project folder
│   ├── src/
│   ├── index.html               # Root HTML (Vite, not in public/)
│   ├── package.json
│   ├── vite.config.js           # Vite config
│   └── ... (all other files)
├── README.md
└── screenshot.png
```

How to Submit

Option 1: Upload to GitHub (Recommended)

```
git init
git add .
git commit -m "Lab 1: React Fundamentals Complete (Vite Edition)"
git branch -M main
git remote add origin https://github.com/yourusername/lab-1-react.git
git push -u origin main
```

- Share the GitHub link

Option 2: Zip and Email

- Compress entire Lab-1-Submission folder
- Email to your instructor

Bonus: Self-Assessment

Before submitting, answer these questions honestly:

1. **Understanding:**
   - ☐ I understand what JSX is
   - ☐ I can create a functional component
   - ☐ I understand how props work
   - ☐ I can compose components together
   - ☐ I know when to use `.jsx` vs `.js`
2. **Skills:**
   - ☐ I can fix JSX syntax errors

- ☐ I can use props to pass data
- ☐ I can use `.map()` to render lists
- ☐ I can style components with CSS
- ☐ I can set up a Vite React project

3. **Confidence:**

- What was easiest? (Component creation, props, styling, composition, Vite setup?)
- What was hardest? (JSX, props, styling, composition, file structure?)
- Do I feel ready for Chapter 2 (State)?
- Did Vite feel fast compared to CRA?

Common Mistakes to Avoid

❌ Don't:

1. **Forget to export components**

```
// ❌ Wrong - no export
function MyComponent() { ... }

// ☑ Right
export default MyComponent
```

2. **Use lowercase for component names**

```
// ❌ Wrong
function greeting() { ... }

// ☑ Right
function Greeting() { ... }
```

3. **Use `.js` extension for React components**

```
❌ Greeting.js
☑ Greeting.jsx
```

4. **Return multiple elements without wrapper**

```
// ❌ Wrong
return (
  <h1>Title</h1>
  <p>Content</p>
)
```

```
// ☑ Right (use <> </>)
return (
  <>
    <h1>Title</h1>
    <p>Content</p>
  </>
)
```

5. **Forget to import components**

```
// ❌ Wrong - didn't import
export default function App() {
  return <UserCard />  // ERROR!
}

// ☑ Right
import UserCard from './components/UserCard'

export default function App() {
  return <UserCard />
}
```

6. **Use `class` instead of `className`**

```
// ❌ Wrong
<div class="container">Content</div>

// ☑ Right
<div className="container">Content</div>
```

7. **Forget to use key prop in lists**

```
// ⚠ Works but not ideal
items.map((item, index) => <Movie key={index} ... />)

// ☑ Better - use unique id
items.map((item) => <Movie key={item.id} ... />)
```

Getting Help

If you get stuck:

1. **Read the error message carefully** - Vite and React give helpful error messages!
2. **Check file extensions** - Make sure React components use `.jsx`

3. **Revisit Chapter 1** - Look for similar examples
4. **Check your imports** - Most errors are missing imports or wrong paths
5. **Inspect in browser DevTools** - Press F12 to check the Elements tab
6. **Use React DevTools** - Inspect components and props
7. **Ask your instructor** - Don't struggle alone!

Next Steps

After completing this lab:

- ☑ Review your code and make sure you understand every line
- ☑ Show your portfolio to someone and get feedback
- ☑ Check your Vite project structure against Chapter 1
- ☑ Read Chapter 2: State & Forms
- ☑ Come back to this lab later and refactor with what you learned

---

Good luck! You've got this! 🚀

Remember: The goal isn't perfection—it's understanding. Every mistake is a learning opportunity.

Welcome to the world of Vite + React! ⚡

Answer Key (For Instructors Only)

▶ Click to expand (Students: Try without this first!)

Exercise 1.1 Solution Example

```jsx
// src/components/Greeting.jsx
import './Greeting.css'

export default function Greeting() {
  return (
    <div className="greeting-container">
      <h1>Welcome to React!</h1>
      <h2>This is my first component</h2>
      <p>
        Fun Fact: React was created by Facebook and is now maintained by Meta
        and the community! Vite is the modern build tool for React apps!
      </p>
    </div>
  )
}
```

Exercise 1.2 Solutions

```jsx
// ERROR 1: Multiple root elements - wrap in div or use <>
function FixedComponent1() {
  return (
    <>
      <h1>Hello</h1>
      <p>This is fixed</p>
    </>
  )
}

// ERROR 2: Can't use if statement inside JSX - use ternary
function FixedComponent2() {
  const isTrue = true
  return (
    <div>
      <p>Result: {isTrue ? 'Yes' : 'No'}</p>
    </div>
  )
}

// ERROR 3: Use className and self-closing tags
function FixedComponent3() {
  return (
    <div className="container">
      <img src="cat.jpg" alt="A cat" />
      <p>A paragraph</p>
    </div>
  )
}
```

Exercise 2.1 Solution Example

```jsx
// src/components/UserCard.jsx
import './UserCard.css'

export default function UserCard({ name, email, role }) {
  return (
    <div className="user-card">
      <h3>{name}</h3>
      <p><strong>Email:</strong> {email}</p>
      <p><strong>Role:</strong> {role}</p>
    </div>
  )
}
```

```jsx
// In App.jsx
import UserCard from './components/UserCard'
```

```jsx
function App() {
  return (
    <div className="app">
      <UserCard
        name="Alice"
        email="alice@example.com"
        role="Developer"
      />
      <UserCard
        name="Bob"
        email="bob@example.com"
        role="Designer"
      />
      <UserCard
        name="Charlie"
        email="charlie@example.com"
        role="Manager"
      />
    </div>
  )
}

export default App
```

Portfolio Challenge - App.jsx Example

```jsx
import Header from './components/Header/Header'
import About from './components/About/About'
import ProjectShowcase from './components/ProjectShowcase/ProjectShowcase'
import Contact from './components/Contact/Contact'
import Footer from './components/Footer/Footer'
import './App.css'

function App() {
  const projects = [
    {
      id: 1,
      name: 'Todo App',
      description: 'A React task management app',
      technologies: ['React', 'JavaScript', 'CSS']
    },
    {
      id: 2,
      name: 'Weather Dashboard',
      description: 'Real-time weather using API',
      technologies: ['React', 'API', 'Charts']
    },
    {
      id: 3,
      name: 'E-commerce Site',
      description: 'Full shopping experience',
```

```
      technologies: ['React', 'Node.js', 'MongoDB']
    }
  ]

  return (
    <div className="portfolio">
      <Header name="Your Name" tagline="Full Stack Developer" />
      <About
        profileImage="https://via.placeholder.com/150"
        bio="I'm passionate about building amazing web experiences..."
        skills={['React', 'JavaScript', 'CSS', 'HTML', 'Node.js']}
      />
      <ProjectShowcase projects={projects} />
      <Contact
        email="your@email.com"
        github="github.com/yourname"
        linkedin="linkedin.com/in/yourname"
      />
      <Footer name="Your Name" />
    </div>
  )
}

export default App
```

Lab 1 Complete! Ready for Chapter 2? 🎓

Remember: You're using Vite (modern, fast) with proper `.jsx` files (professional, clear)! ⚡