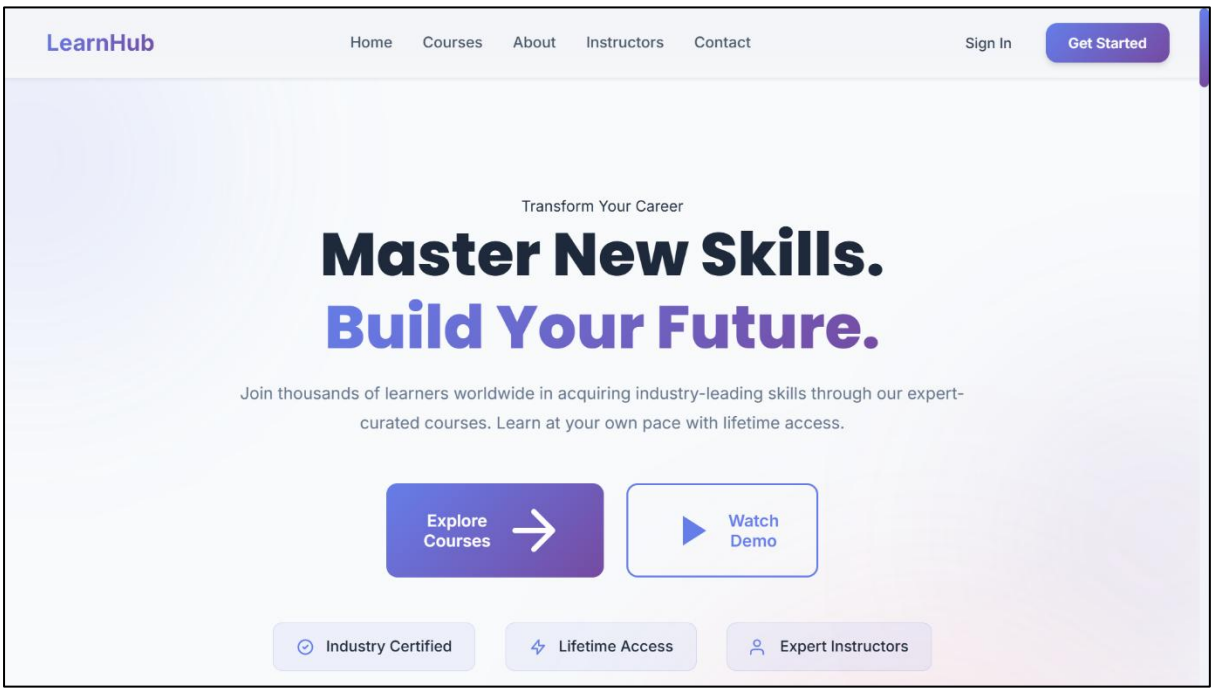
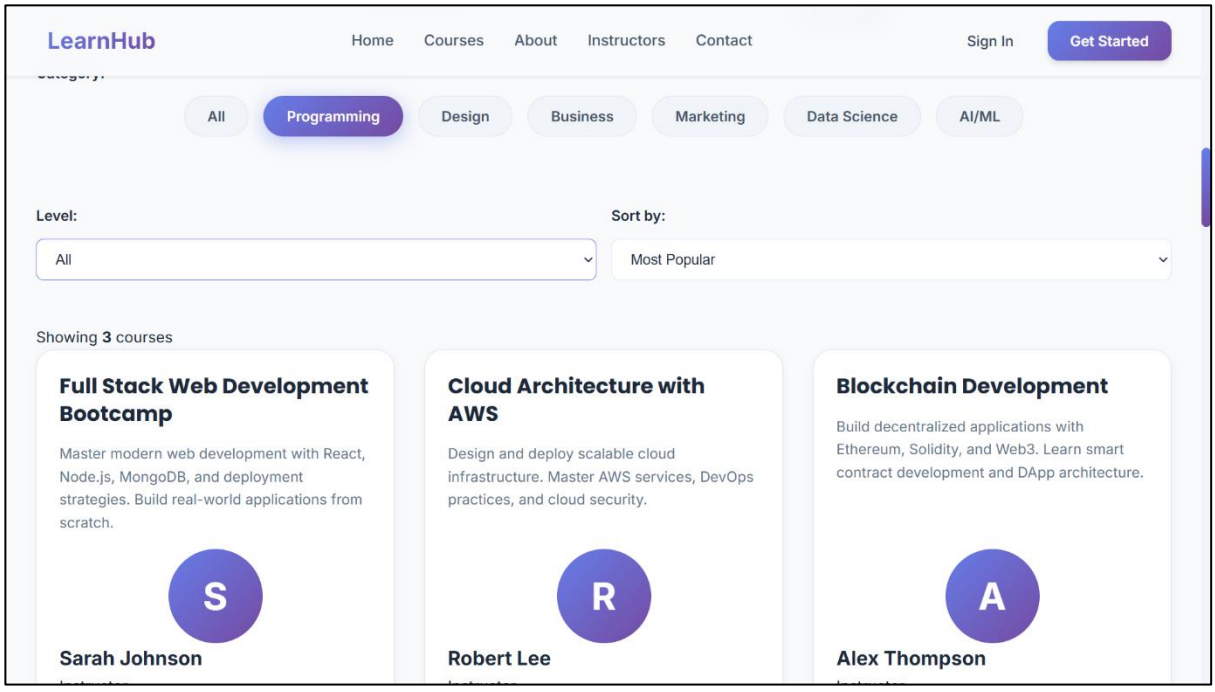
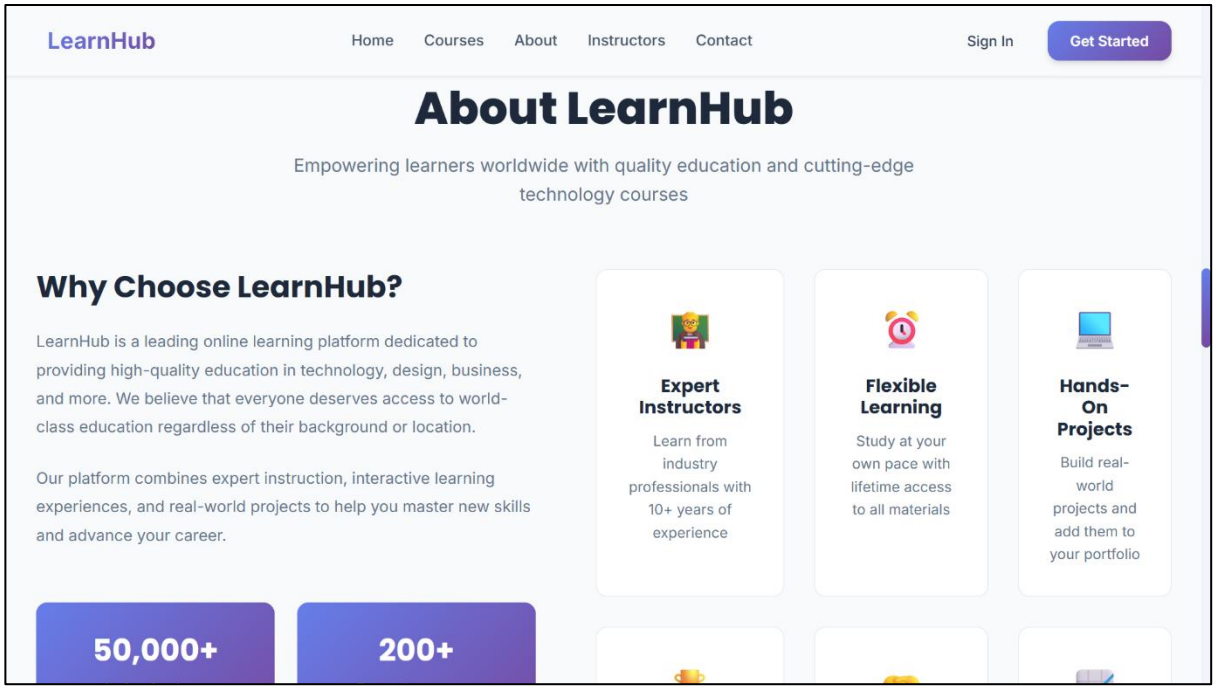
DEVOPS AND FULLSTACK

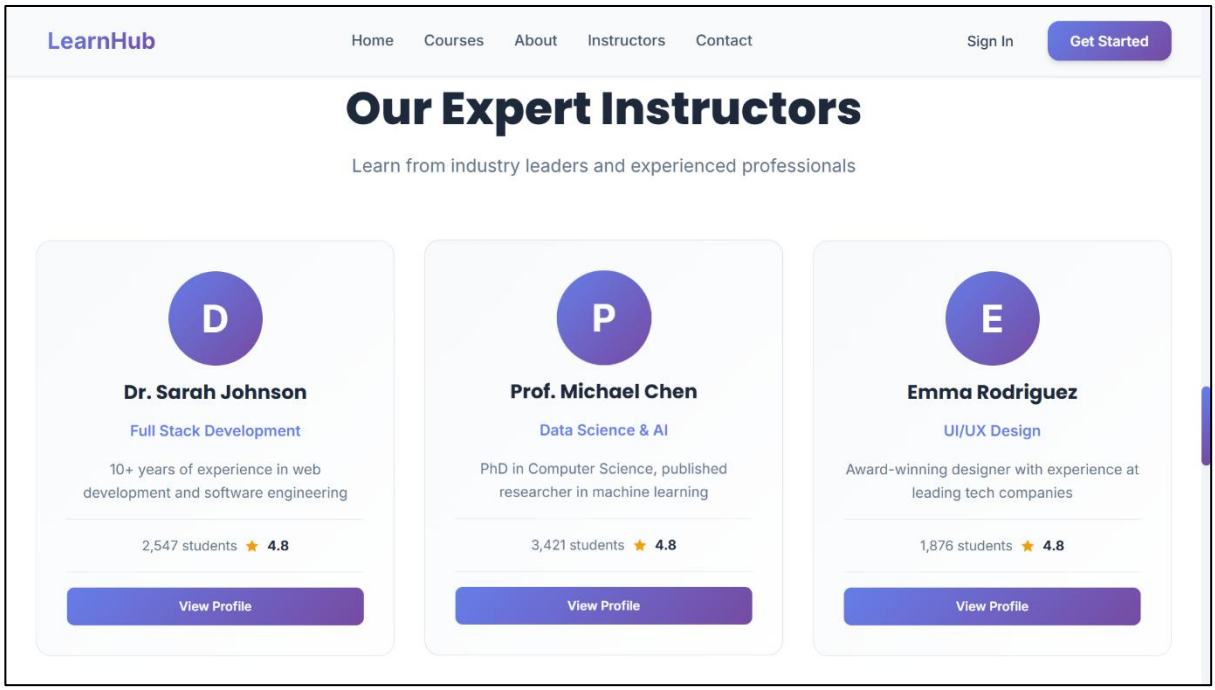
Name: Lalu prasad Aroori
Roll No.: 2303A51948
Date: 28-01-2026
Batch: 07
Assignment Number: 8/24(week 4) Wednesday

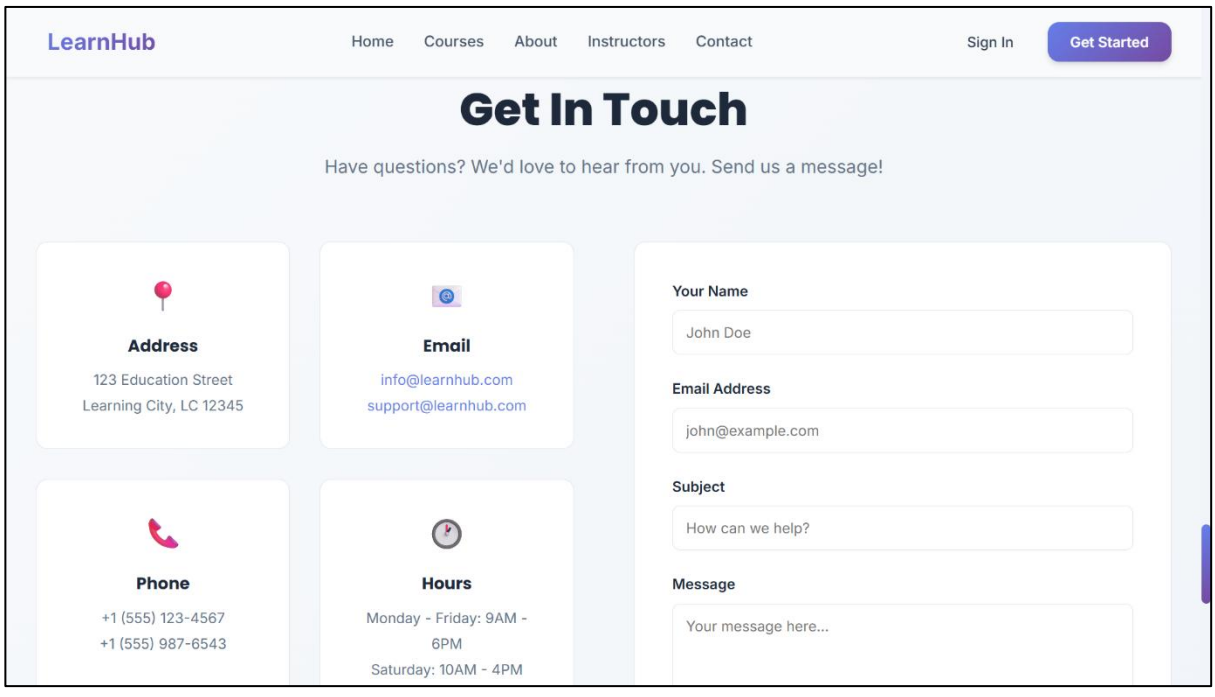
Results:






LearnHub
Home
Courses
About
Instructors
Contact
Sign In
Get Started
About LearnHub
Empowering learners worldwide with quality education and cutting-edge technology courses
Why Choose LearnHub?
LearnHub is a leading online learning platform dedicated to providing high-quality education in technology, design, business, and more. We believe that everyone deserves access to world-class education regardless of their background or location.
Our platform combines expert instruction, interactive learning experiences, and real-world projects to help you master new skills and advance your career.
Expert Instructors
Learn from industry professionals with 10+ years of experience
Flexible Learning
Study at your own pace with lifetime access to all materials
Hands-On Projects
Build real-world projects and add them to your portfolio
50,000+
200+


LearnHub
Home
Courses
About
Instructors
Contact
Sign In
Get Started
Our Expert Instructors
Learn from industry leaders and experienced professionals
D
Dr. Sarah Johnson
Full Stack Development
10+ years of experience in web development and software engineering
2,547 students
4.8
View Profile
P
Prof. Michael Chen
Data Science & AI
PhD in Computer Science, published researcher in machine learning
3,421 students
4.8
View Profile
E
Emma Rodriguez
UI/UX Design
Award-winning designer with experience at leading tech companies
1,876 students
4.8
View Profile


LearnHub
Home
Courses
About
Instructors
Contact
Sign In
Get Started
Get In Touch
Have questions? We'd love to hear from you. Send us a message!
Address
123 Education Street
Learning City, LC 12345
Email
info@learnhub.com
support@learnhub.com
Phone
+1 (555) 123-4567
+1 (555) 987-6543
Hours
Monday - Friday: 9AM - 6PM
Saturday: 10AM - 4PM
Your Name
John Doe
Email Address
john@example.com
Subject
How can we help?
Message
Your message here...

React Questions and Answers

Based on Course Dashboard Project Implementation

Question 1: How would you set up a new React project using tools like Create React App or Vite?

In our project, we used **Vite** to set up the React application. Here's how we did it:

```
npm create vite@latest course-dashboard -- --template react
cd course-dashboard
npm install
npm run dev
```

Why we chose Vite over Create React App:

- **Lightning fast** - Vite starts the dev server instantly, while CRA can take 20-30 seconds
- **Hot Module Replacement (HMR)** - Changes appear immediately in the browser without full page reload
- **Modern build tool** - Uses native ES modules and esbuild for faster builds **Smaller**
- **bundle size** - Vite produces more optimized production builds

In our `vite.config.js` , we configured the server port and proxy settings:

```
export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      '/api': 'http://localhost:5000'
    }
  }
})
```

Question 2: What is the role of package.json in a React project?

The `package.json` file is like the **blueprint** of our project. In our course dashboard, it serves several critical roles:

1. Dependency Management:

```
"dependencies": {
  "react": "^19.0.0",
  "react-dom": "^19.0.0",
  "axios": "^1.7.9"
}
```

This tells npm exactly which libraries our project needs. When someone clones our repository, they just run `npm install` and all these packages get installed automatically.

2. Scripts for Development:

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
}
```

These scripts let us run commands like `npm run dev` to start the development server or `npm run build` to create the production-ready version.

3. Project Metadata:

```
{
  "name": "course-dashboard",
  "private": true,
  "version": "0.0.0",
  "type": "module"
}
```

This identifies our project and specifies we're using ES modules.

Real-world example: When we added Axios for API calls, we ran `npm install axios` , which automatically updated our `package.json` and `package-lock.json` files.

Question 3: How do you create a functional component in React?

In our project, we created **8 functional components**. Here's how we built them:

Simple Component Example - Header:

```
const Header = () => {
  return (
    <header className="header">
      <div className="header-container">
        <a href="#" className="logo">LearnHub</a>
        <nav className="nav">
          <a href="#home">Home</a>
          <a href="#courses">Courses</a>
          <a href="#about">About</a>
        </nav>
      </div>
    </header>
  );
};

export default Header;
```

Component with Props - CourseCard:

```
const CourseCard = ({ course }) => {
  return (
    <div className="course-card">
      <div className="course-content">
        <h3 className="course-title">{course.title}</h3>
        <p className="course-description">{course.description}</p>
        <div className="course-stats">
          <span className="stat-value">{course.students}+ students</span>
          <span className="course-price">${course.price}</span>
        </div>
      </div>
    </div>
  );
};

export default CourseCard;
```

Component with State - CoursesGrid:

```
const CoursesGrid = () => {
  const [courses, setCourses] = useState([]);
  const [loading, setLoading] = useState(true);
```

```
  const [searchQuery, setSearchQuery] = useState('');

  useEffect(() => {
    fetchCourses();
  }, []);

  return (
    <section className="courses-section">
      {/* Component JSX */}
    </section>
  );
};

export default CoursesGrid;
```

Key characteristics of functional components:

- Use arrow functions or regular functions
- Return JSX (looks like HTML but it's JavaScript)
- Can use Hooks like `useState` and `useEffect`
- Must start with a capital letter (`Header` , not `header`) Export
- at the end so other files can import them

Question 4: How are components rendered inside the main App component?

In our project, we use a **hierarchical component structure**. Here's how it works:

Entry Point - main.jsx:

```
import App from './App.jsx'

ReactDOM.createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

This tells React to render our `App` component inside the HTML element with id="root".

Main App Component - App.jsx:

```
function App() {
  return (
```

```
    <div className="App">
      <Dashboard />
    </div>
  );
}

export default App;
```

Simple and clean - our App just renders the Dashboard component.

Dashboard Component - Dashboard.jsx:

```
const Dashboard = () => {
  return (
    <div className="dashboard">
      <Header />
      <main className="main-content">
        <WelcomeSection />
        <CoursesGrid />
        <About />
        <Instructors />
        <Contact />
      </main>
      <Footer />
    </div>
  );
};

export default Dashboard;
```

The Component Tree:

```
App └── Dashboard ├── Header ├── main │ ├── WelcomeSection │ ├── CoursesGrid │ │ └── CourseCard (rendered multiple times) │ ├── About │ ├── Instructors │ └── Contact └── Footer
```

How data flows:

- Dashboard doesn't need to know the internal details of each component
- Each component is self-contained with its own styling and logic
- Props flow downward: `CoursesGrid` passes course data to each `CourseCard`
- Components can be reordered easily (we can move `About` above `CoursesGrid` without breaking anything)

Question 5: What are the benefits of breaking the UI into small reusable components?

From building this course dashboard, here are the **real benefits** we experienced:

1. Reusability:

```
// CourseCard component is used 8 times (once for each course)
{courses.map((course) => (
  <CourseCard key={course.id} course={course} />
))}
```

We wrote the card design once and reused it for all courses. If we want to change how cards look, we edit **one file**, and all 8 cards update automatically.

2. Maintainability:

When asked to "hide the category badge and fix stats visibility," we only edited `CourseCard.css` :

```
.course-category-badge {
  display: none;
}
```

We didn't need to touch 8 different places. One change = fixed everywhere.

3. Easier Debugging:

When the header wasn't sticking properly, we knew exactly where to look - `Header.css` . We didn't have to search through thousands of lines of code.

4. Team Collaboration:

In a real team:

- Designer A can work on `Header.jsx` and `Header.css`
- Designer B can work on `Footer.jsx` and `Footer.css` Developer C
- can work on `CoursesGrid.jsx`

They won't step on each other's toes because components are isolated.

5. Testing Made Simple:

You can test each component independently:

```
// Test CourseCard alone
<CourseCard course={mockCourse} />

// Test CoursesGrid with fake data
<CoursesGrid courses={testCourses} />
```

6. Easy to Add Features:

When About, Instructors, and Contact sections were needed, we just:

1. Created three new components
2. Added them to Dashboard
3. Styled them independently

No existing code was affected. The header, footer, and courses sections kept working perfectly.

7. Better Performance:

React only re-renders components that change. If a user types in the search bar (in `CoursesGrid`), React doesn't re-render the `Header` or `Footer` - only the courses list updates.

8. Code Organization:

Our project structure is clean:

```
components/ ├── Header.jsx ├── WelcomeSection.jsx ├──
CoursesGrid.jsx ├── CourseCard.jsx ├── About.jsx ├──
Instructors.jsx ├── Contact.jsx └── Footer.jsx
```

Anyone can find what they need in seconds.

Real-world analogy: Think of components like LEGO blocks. You can build a castle using the same brick component 100 times. If you want all red bricks to be blue, you change one brick mold, not 100 individual bricks. That's the power of component-based architecture.
