React Advanced Concepts – Online Learning Platform

Introduction

This project demonstrates the implementation of **advanced React concepts** by building an **Online Learning Platform**.

The main objective of the application is to improve **performance, reliability, and user experience** using real-world React patterns.

The following advanced concepts are implemented:

- Lazy Loading & Code Splitting
- Pure Components
- Error Boundaries
- React Portals

Each concept is implemented based on a user story and enhanced with bonus features for better demonstration and usability.

Challenge 1: Lazy Loading & Code Splitting

User Story

As a user of an online learning platform, I want modules to load only when I click on them so that the application loads faster.

Implementation

Lazy loading is implemented using **React.lazy()** and **Suspense** to dynamically import the **CourseDetails** and **InstructorProfile** components.
These components are loaded only when the user interacts with the respective buttons, reducing the initial bundle size.

To clearly demonstrate the loading behavior during evaluation, an **artificial delay of 500 ms** is introduced while importing the components. This ensures the loading spinner is visible and easy to capture.

Code Snippet 1: Lazy loading with artificial delay

```
const lazyWithDelay = (importFn, delay = 500) => {
  return lazy(() =>
    Promise.all([
      importFn(),
      new Promise(resolve => setTimeout(resolve, delay))
    ]).then(([module]) => module)
  );
};
```

Code Snippet 2: Suspense with loader

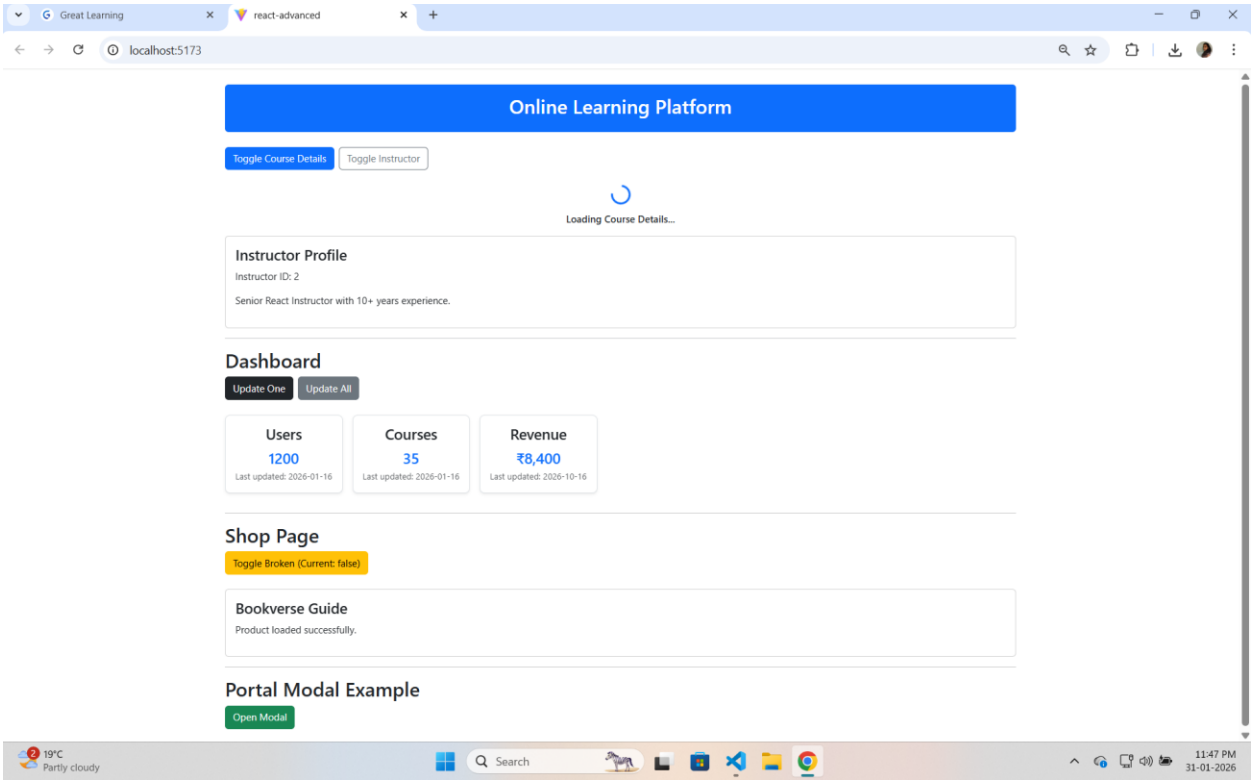
```
<Suspense fallback={<Loader message="Loading Course Details..." />}>
  <CourseDetails courseId={1} />
</Suspense>
```

Bonus Features Implemented

- Bootstrap spinner during loading
- Context-specific loading messages
- Artificial delay for demonstration purposes

Output Screenshot

Figure 1: Lazy loading with loading spinner



Challenge 2: Pure Components

User Story

As a dashboard viewer, I want widgets to re-render only when their data changes so that performance is optimized.

Implementation

The **StatsCard** component is implemented as a pure component using **React.memo()**.
This prevents unnecessary re-renders by performing a shallow comparison of props.

A **Dashboard page** displays multiple statistic cards and includes buttons to simulate updates.

Console logs are used to verify render behavior.

Code Snippet: Pure Component using React.memo

```
const StatsCard = React.memo(function StatsCardInner({ title, value }) {
```

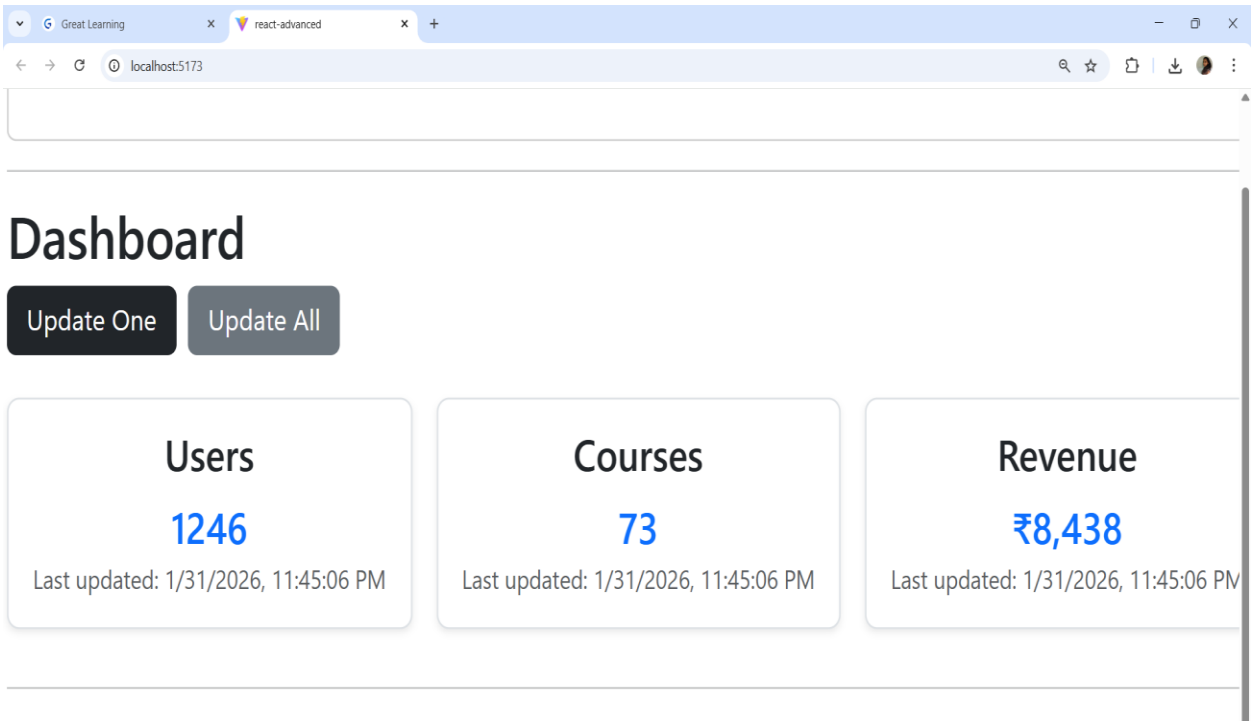
```
  console.log(`Render: ${title}`);
  return <div>{value}</div>;
});
```

Bonus Features Implemented

- “Update One” button to update a single card
- “Update All” button to update all cards
- Console logging to observe optimization

Output Screenshot

Figure 2: Dashboard demonstrating Pure Component behavior



Challenge 3: Error Boundary

User Story

As a product manager, I want the app to display a friendly error message instead of crashing when a component fails.

Implementation

An **Error Boundary** is implemented as a class component using:

- getDerivedStateFromError()
- componentDidCatch()

A broken component (**BrokenProductCard**) is intentionally used to trigger a runtime error. The Error Boundary catches the error and displays a fallback UI without breaking the rest of the application.

Code Snippet: Error Boundary lifecycle methods

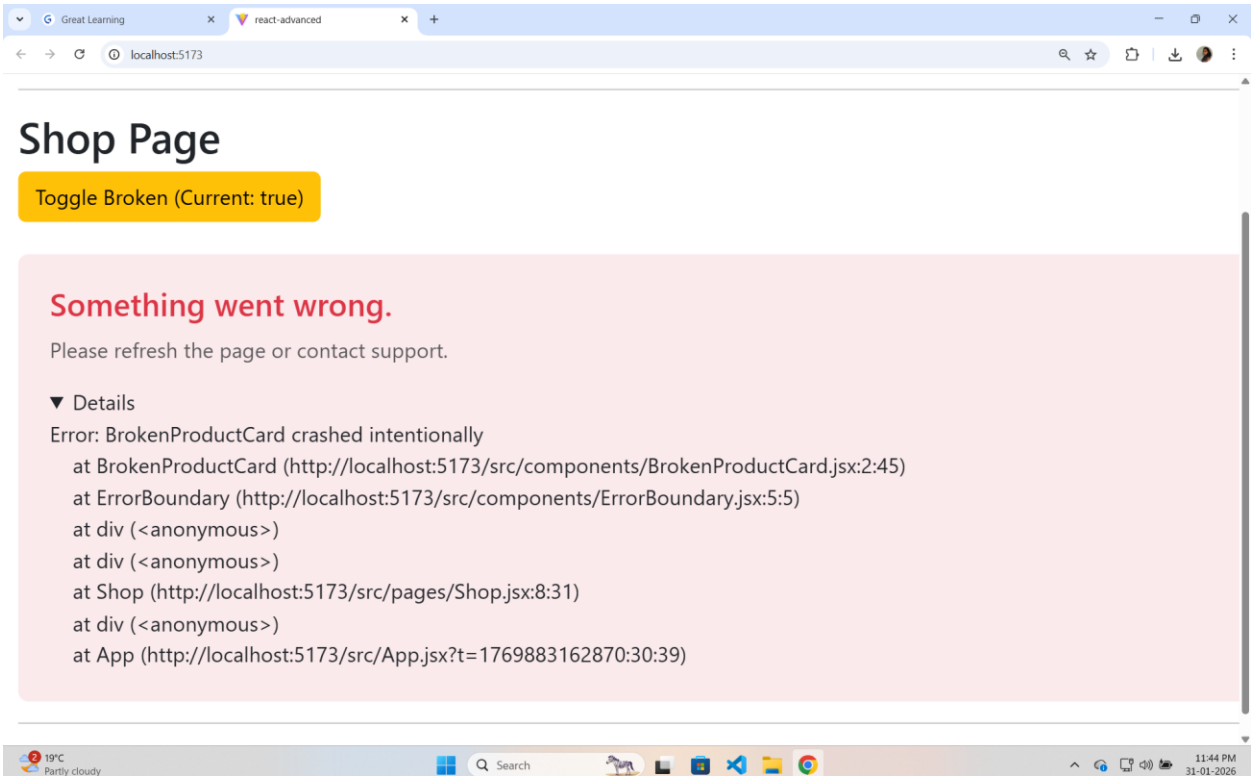
```
static getDerivedStateFromError(error) {
  return { hasError: true };
}

componentDidCatch(error, info) {
  console.error("Error caught:", error, info);
}
```

Bonus Features Implemented

- Error logging to the console
- Display of error details using <details>
- User-friendly fallback UI

Output Screenshot

Figure 3: Error Boundary



Challenge 4: React Portals

User Story

As a user, I want notifications and modal popups to appear above all UI elements, even if nested inside components.

Implementation

React Portals are used to render a modal into a separate DOM node (modal-root) outside the main React hierarchy using **ReactDOM.createPortal()**.

The modal supports:

- Open and close using state
- Backdrop click to close
- Escape key to close

Code Snippet: Portal rendering

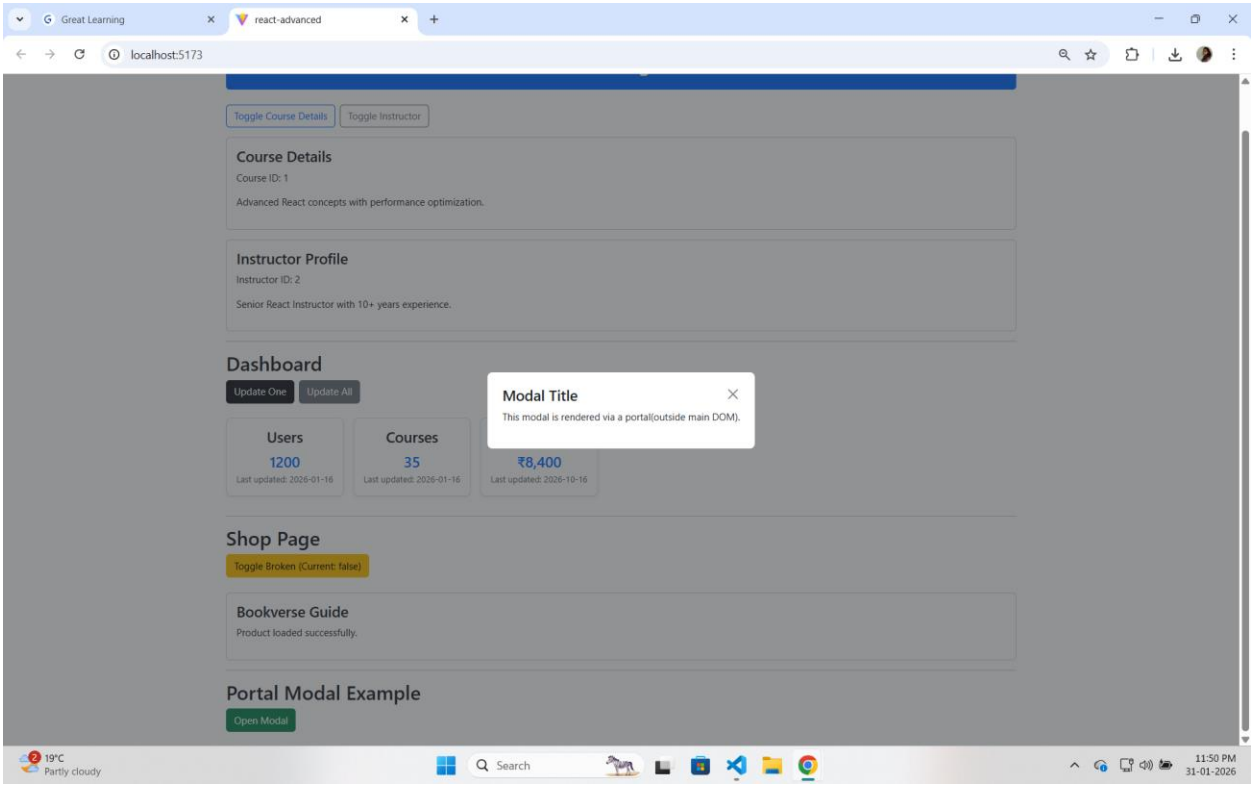
```
ReactDOM.createPortal(
  <div className="modal-overlay">...</div>,
  document.getElementById("modal-root")
);
```

Bonus Features Implemented

- Keyboard accessibility
- Clean overlay styling using Bootstrap

Output Screenshot

Figure 4: Modal rendered using React Portal outside the main DOM hierarchy

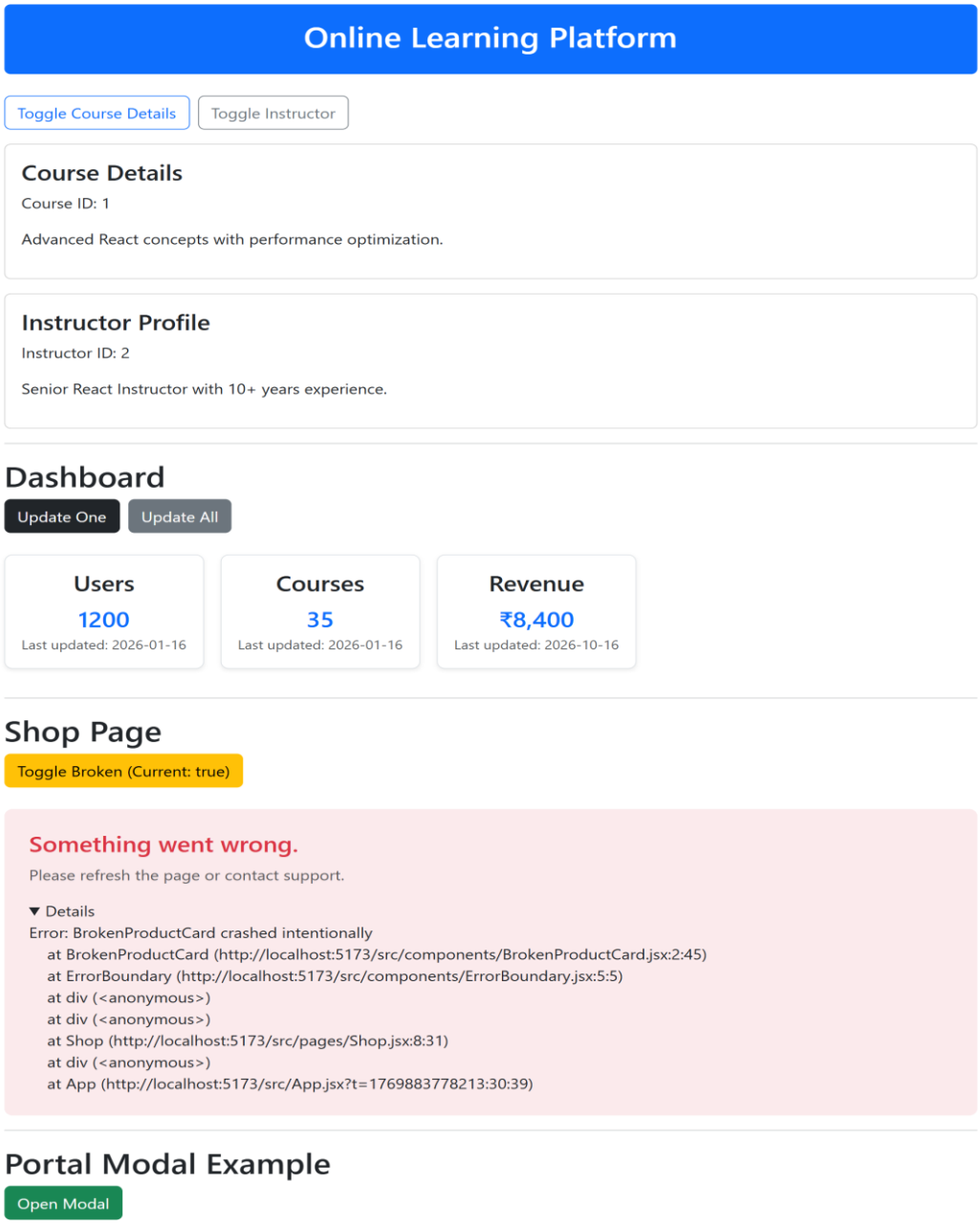


Complete Application View

This section shows the fully integrated Online Learning Platform containing all implemented features.

Output Screenshot

Figure 5: Complete Online Learning Platform showcasing all implemented features



Conclusion

This project successfully demonstrates advanced React concepts used in real-world applications. By implementing lazy loading, pure components, error boundaries, and portals, the application achieves:

- Improved performance
- Enhanced reliability
- Better user experience