Proof of Concept Report: Migrating C# Razor Frontend to React.js

**Course**: Transactional Web Applications  
**Project Title**: Comparative Analysis of Deployment Strategies and Architectures for an E-Commerce Platform  
**Student**: Reza Akbari  
**Date**: 04/07/2025

## 1. Project Title

Modernizing IdealShop Frontend with React: A Comparative Proof of Concept

## 2. Objective & Business Need

The project aims to enhance user experience and front-end scalability of the IdealShop e-commerce platform. Originally built with ASP.NET Core MVC and Razor Views, the frontend lacked modern interactivity and maintainability. Given evolving user expectations and the rise of component-based frameworks, we replaced the Razor frontend with React.js and integrated it with the existing ASP.NET Core Web API backend.

## 3. Business Scenario

IdealShop is an online store allowing customers to browse products, register/login, manage carts, and place orders. Admins manage categories, products, and users. The legacy system used Razor Pages, tightly coupled to backend logic, which limited the frontend's interactivity and maintainability. To modernize the application, a React-based SPA frontend was developed, connected via RESTful APIs to the backend.

## 4. Hypotheses

1. React.js will offer faster frontend development and UI flexibility compared to Razor Views.

2. Separating frontend (React) from backend (ASP.NET Core) improves maintainability and scalability.

3. The new architecture allows for better client-side performance, especially in dynamic interactions like cart and authentication.

## 5. Technology Comparison (Stacks)

Legacy Stack:

- Frontend: C# Razor Pages

- Backend: ASP.NET Core API

- Middleware: N/A

- Database: SQL Server

- Authentication: Cookie Auth (.NET)

New Stack:

- Frontend: React.js

- Backend: ASP.NET Core API

- Middleware: Axios + Cookies

- Database: SQL Server

- Authentication: Cookie Auth

## 6. Roadmap

1. Extract frontend from Razor.

2. Build UI components using React.

3. Connect frontend with backend via Axios and REST APIs.

4. Replace server-side rendering with client-side routing.

5. Handle customer and admin flows in React.

6. Secure communication using cookies.

## 7. Prototype Features

- Customer: Register/Login, Browse Products, Product Details, Add to Cart, Checkout

- Admin: Login, Manage Categories, Products, Admins, Customers

- Shared: Dynamic Navbar, Authentication-aware features

- Tested via Postman and browser tools

## 8. Results & Evaluation

React provided:

- Faster UI development

- Better user experience

- Clear code separation

- Better performance for dynamic UI tasks like cart

Confirmed Hypotheses:

✔ UI development is faster

✔ Architecture is more scalable

✔ Interactions are more responsive

## 9. Challenges

- Syncing auth state between backend and React

- Cookie-based auth config with Axios

- Identity not always available without credentials

- UI required localStorage and state checks

## 10. Conclusion

React proved more maintainable and interactive than Razor Pages. The new structure matches modern frontend expectations. Our goals were met and the shift was successful.

## 11. Next Steps

- Containerize both apps using Docker

- Deploy frontend to Vercel or Netlify, backend to Azure

- Conduct performance and load testing with Lighthouse and Postman