**Proof of Concept: Student Course Registration System**

**Comparative Analysis: Razor Pages (.NET) vs React + Node.js**

**Objective**

To evaluate and compare two different technology stacks for building a student course registration web application, focusing on **functionality, scalability, security, ease of development, and system integration**. This POC helps determine which stack offers the best balance between business needs, performance, and developer agility.

**Stacks Overview**

| **Stack** | **Vishnu (Team A)** | **Danila (Team B)** |
| --- | --- | --- |
| Frontend | Razor Pages (ASP.NET Core) | React.js |
| Backend | .NET Core PageModel | Node.js (Express) |
| Database | Microsoft SQL Server | MySQL |
| Language | C# | JavaScript |
| Architecture | Layered MVC using Razor | RESTful API (frontend-backend decoupled) |
| Hosting | IIS / Azure | Node + Apache/Nginx / AWS |
| Security | Built-in Identity + 2FA | JWT + OAuth (optional) |

**Hypotheses & Evaluation Criteria**

We will compare both stacks on the following 3 axes:

**1. Ease of Integration & Architecture**

| **Criteria** | **Razor Pages** | **React + Node** |
| --- | --- | --- |
| Tight integration between frontend and backend | (page-centric model) | (requires explicit API layer) |
| Decoupling for scalability | Moderate | (frontend/backend fully separate) |
| Routing | Handled via Razor conventions | Custom routing with React Router & Express |
| API flexibility | Less flexible for mobile apps | Reusable REST APIs |

**Winner: React + Node** (for long-term flexibility)

**2. Security Measures**

| **Criteria** | **Razor Pages** | **React + Node** |
| --- | --- | --- |
| Built-in Identity, login, 2FA | Robust with ASP.NET Identity | Needs custom implementation |
| SQL Injection prevention | ORM + Parameterized Queries | ORM (Sequelize/Knex) or manual queries |
| XSS Protection | Razor auto-escapes | Requires explicit escaping |
| HTTPS + SSL | Self-made SSL tested locally | Must be configured (Let’s Encrypt, etc.) |

**Winner: Razor Pages** (faster to implement secure login flows)

**3. Developer Productivity & Maintenance**

| **Criteria** | **Razor Pages** | **React + Node** |
| --- | --- | --- |
| Rapid prototyping | with Visual Studio | Slower, requires setup |
| Ecosystem & packages | .NET packages available | Larger NPM ecosystem |
| Maintainability | Centralized Razor logic | Split logic (JS frontend, Node backend) |
| Team Familiarity (from reports) | C# + SQL | React + Node.js |

**Winner: Razor Pages** (for small-to-medium projects, solo devs)

**Prototype Implementation Summary**

**Team A (Razor Pages + MSSQL):**

* Complete working prototype with login, registration, security measures
* XSS + SQL injection intentionally added for educational vulnerability testing
* UI built with Razor Pages and PageModel

**Team B (React + NodeJS + MySQL):**

* Frontend developed with React, backend with NodeJS
* Uses Express for routing, Axios for API calls
* Data stored and queried from MySQL

**Results & Feedback**

* **Razor stack** provides faster initial development and built-in security tools, making it ideal for solo developers and internal tools.
* **React + Node** offers better scalability, cross-platform integration (e.g., mobile apps), and team collaboration capabilities with clear API boundaries.

**Recommendation**

* For **faster MVPs or internal college systems**, go with **Razor Pages + MSSQL**.
* For **enterprise-grade systems with future mobile expansion**, choose **React + NodeJS + MySQL**.