Version & Metadata

Version: v1.1
Title: PeerLearn — Tech Stack Analysis
Project: PeerLearn (Personal project)
Author: Aleksandar Ivanov
Date: 2025-09-11 (Europe/Amsterdam)
Status: Draft for review
Change Log:

- v1.1 — Added explicit references/links and SQL vs MongoDB section with a hybrid plan; aligned with university constraints.
- v1.0 — Initial stack rationale (C#/.NET + React).

1) Purpose

This document explains **why** PeerLearn uses **C# / ASP.NET Core** for the backend and **React** for the frontend, and how we'll reconcile the university's **SQL expectation** with a developer preference for **MongoDB**. All non-project claims are linked to primary sources.

Constraints provided by course:

- Backend must be C#/.NET (or Razor Pages); author prefers Web API + React. *(project constraint)*
- University expects use of SQL. *(project constraint)*

2) Backend Choice — C# / ASP.NET Core Web API

Why ASP.NET Core now?

- **Stable support window:** .NET 8 is the current **LTS** (supported until **Nov 10, 2026**); .NET 9 is **STS** (through **May 12, 2026**). This lets us target .NET 8 during the 2025 academic cycle with a clean upgrade path. Microsoft support policy Lifecycle table What's new in .NET 9
- **Designed for Web APIs:** First-class patterns for controllers and **Minimal APIs**; choice can be pragmatic per endpoint. Web API overview Minimal APIs
- **Real-time built-in: SignalR** provides a batteries-included real-time abstraction over WebSockets and fallbacks, with official **JavaScript client** for the React app. SignalR intro JS client @microsoft/signalr
- **Security & platform features:** built-in **Data Protection** (key storage/rotation) and **RateLimiting** middleware. Data Protection config Key lifetime Rate limiting middleware
- **Ecosystem & performance visibility:** ASP.NET Core appears in the **TechEmpower** benchmarks rounds, making performance tuning easier to contextualize over time.

(No single benchmark determines real-world perf; we measure our own endpoints.) TechEmpower Benchmarks

Auth stance:

- Use **JWT** access tokens validated on every request (`iss`,`aud`,`exp`,`nbf`,`iat`). OWASP REST Security
- Prefer delivery via **HttpOnly** cookies to reduce token theft via XSS; avoid `localStorage` for session identifiers. OWASP HttpOnly HTML5 Security Cheat Sheet

3) Frontend Choice — React

- **Adoption & workforce familiarity:** 2025 Stack Overflow survey shows **React** used by ~44.7% of all respondents and ~46.9% of professional developers in the "Web frameworks & technologies" category. SO Survey 2025
- **Editor ecosystem:** React pairs well with **Slate** for building a Notion-style rich-text editor. Slate docs
- **Collab ecosystem:** React integrates cleanly with **Yjs** providers (`y-websocket` recommended for central auth). Yjs docs y-websocket

4) Database Strategy — Reconciling SQL Expectation with MongoDB Preference

What the course expects: Use SQL somewhere in the project. *(project constraint)*

What the app needs: Flexible document-style storage for collaborative **notes**, which change in small operations and benefit from schema flexibility and CRDT snapshots. **MongoDB** is a fit for this workload. MongoDB .NET driver Yjs docs

How to satisfy both (hybrid approach):

- Store **content** (notes, flashcards, quiz results) in **MongoDB**. MongoDB .NET driver
- Store **"compliance-friendly" tables** (users/audit/logs, or even a subset of room metadata) in **SQL** using **EF Core** (SQL Server or PostgreSQL). This proves competence with relational modeling and ACID transactions. EF Core providers overview SQL Server provider Npgsql EF Core ACID reference (PostgreSQL)

Why document DB for notes?

- Notes are nested JSON-like data updated via collaborative ops. Document stores avoid costly impedance mismatch and allow **schema-on-read** patterns. Non-relational data (Azure guide)

Optional convenience:

- If you want a single ORM-style programming model across both, an **EF Core provider for MongoDB** exists. It's optional; the official **MongoDB .NET driver** is the proven baseline. MongoDB EF Core provider docs MongoDB .NET driver

5) Real-Time Collaboration Stack

- **Editor: Slate** for the UI/UX layer. Slate docs
- **CRDT: Yjs** for conflict-free, offline-tolerant syncing. Yjs docs
- **Transport:** Prefer **`y-websocket`** (central auth, headers/cookies) over `y-webrtc` for MVP. y-websocket docs y-webrtc repo
- **Presence & events: SignalR** hub complements Yjs for presence/chat/room events. SignalR intro

6) Voice (Post-MVP)

- **WebRTC** for audio streams; always plan for **TURN** to handle strict NAT. WebRTC API (MDN) TURN (MDN) coturn project

7) Risks & Trade-offs

- **Two datastores = operational overhead.** Mitigate with clear ownership per domain and separate connection layers. *(engineering practice)*
- **SQL/NoSQL data divergence.** Keep SQL for audit/users and MongoDB for content; avoid duplicating hot write paths. *(engineering practice)*
- **Security foot-guns.** Poor token storage or missing claim checks can burn you; prefer HttpOnly cookies, short TTLs, strict JWT validation. OWASP HttpOnly REST Security

8) Summary Decision Matrix

- **Backend:** ASP.NET Core Web API on **.NET 8 LTS** for stability in 2025. Support policy
- **Frontend: React** for familiarity and market presence. SO Survey 2025
- **Collab core: Slate + Yjs + y-websocket**; SignalR for presence/room events. Slate Yjs y-websocket SignalR
- **Data: MongoDB** for content + **SQL** for course-required relational tables (EF Core). MongoDB .NET driver EF Core SQL Server Npgsql EF Core

9) References (direct links)

- .NET support & lifecycle: https://dotnet.microsoft.com/en-us/platform/support/policy • https://learn.microsoft.com/en-us/lifecycle/products/microsoft-net-and-net-core • https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-9/overview
- Web API patterns: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-9.0 • Minimal APIs: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/overview?view=aspnetcore-9.0

- SignalR: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-9.0 • JS client: https://learn.microsoft.com/en-us/aspnet/core/signalr/javascript-client?view=aspnetcore-9.0 • https://www.npmjs.com/package/%40microsoft/signalr
- React usage stats: https://survey.stackoverflow.co/2025/technology
- Slate: https://docs.slatejs.org
- Yjs & providers: https://docs.yjs.dev • https://docs.yjs.dev/ecosystem/connection-provider/y-websocket • https://github.com/yjs/y-webrtc
- WebRTC & TURN: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API • https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols#turn • https://github.com/coturn/coturn
- MongoDB: https://www.mongodb.com/docs/drivers/csharp/ • EF Core provider for MongoDB: https://www.mongodb.com/docs/efcore/current/
- EF Core providers (SQL): https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.sqlserver/ • https://www.npgsql.org/efcore/ • https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli
- Non-relational guidance (schema-on-read): https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data
- Benchmarks context: https://www.techempower.com/benchmarks/
- Security (OWASP): REST Security https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html • HttpOnly https://owasp.org/www-community/HttpOnly • HTML5 Security (localStorage) https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html