
Prompt 1: Describe your app and key features

SkillSnap is a full-stack portfolio and project management application designed for developers to showcase their technical skills, highlight projects, and manage a live personal profile. Built with **ASP.NET Core Web API** and **Blazor WebAssembly**, it enables seamless client-server interaction while maintaining a clean user experience.

Key Features:

- **User Authentication & Registration** using ASP.NET Identity and JWT
- **Portfolio Management** including projects and skill tags
- **Role-Based Authorization**, with Admin users having elevated privileges
- **In-Memory Caching** to optimize API performance
- **Persistent Blazor State Management** to track sessions, editing contexts, and user roles
- **Clean UI Components** for displaying profiles, project cards, and skill tags

---

Prompt 2: Discuss development challenges

Several challenges arose during development:

1. **JWT Integration with Blazor** — Ensuring secure token storage and transmission between the client and API, including dynamic updates to HTTP headers based on login/logout events.
2. **EF Core Relationships** — Designing efficient entity relationships between PortfolioUser, Projects, and Skills, while preventing circular references and performance bottlenecks.
3. **State Management in Blazor** — Since Blazor WebAssembly apps don't maintain server sessions, building a scoped UserSessionService was essential for tracking user context across components.
4. **Security Without Server-Side Blazor** — Achieving secure client-server boundaries without the conveniences of server-side rendering or server sessions required careful token handling and API guarding.

Prompt 3: How did you structure business logic, data persistence, and state management?

Business Logic:

- Business rules and authorization policies are encapsulated in API controllers (e.g., [Authorize(Roles = "Admin")]), keeping the Blazor client clean and focused on presentation.

Data Persistence:

- EF Core with a SkillSnapContext handles all database interactions.
- Identity-backed ApplicationUser is integrated into the same context for unified user and app data management.
- SQLite is used for lightweight persistence during development.

State Management:

- UserSessionService is registered as a scoped service in Blazor to persist user info, roles, and editing states across components.
- JWT tokens are stored securely in localStorage and automatically injected into HTTP headers for protected API calls.

Prompt 4: How did you implement security?

SkillSnap uses a layered approach to security:

1. **User Authentication**:
    - ASP.NET Identity handles user creation, password hashing, and credential validation.
    - JWTs are generated on login and sent with every request to protected endpoints.
2. **Authorization**:
    - Role-based authorization ([Authorize(Roles = "Admin")]) is used to restrict access to sensitive API operations like POSTs or updates.
    - The client decodes JWTs to display role-based UI (e.g., show admin tools only if role = "Admin").
3. **Token Handling**:
    - Tokens are stored in local storage and attached via HTTP headers using AuthService.
    - On logout, tokens are purged and headers are cleared to prevent session spoofing.

Prompt 5: What performance improvements did you apply?

Performance optimization was a key part of the architecture:

1. **In-Memory Caching**:
    - Common queries (GET /api/projects, GET /api/skills) use IMemoryCache to avoid unnecessary database hits.
    - Caches are invalidated on any write operation to ensure freshness.
2. **EF Core Optimizations**:
    - .AsNoTracking() is used for read-only queries to reduce EF's memory overhead.
    - .Include() eager loads related entities, avoiding inefficient N+1 query patterns.
3. **Blazor Session Efficiency**:
    - Avoided re-fetching data by tracking state within UserSessionService, minimizing redundant API calls and enhancing perceived performance.
