Non-functional Requirements and Architectural Solutions

The following architectural structure ensures that MusicMatch efficiently handles non-functional requirements, providing a reliable and enjoyable user experience while maintaining maintainable and secure code.

1. Responsiveness

Requirements

- Search operations for songs, artists, and events must return results within 2 seconds

- Playlist updates (add/remove songs) must be reflected within 1 second

- Chat messages must appear in real-time with maximum 1-second latency

- Profile and event history loading must complete within 3 seconds

- Notifications for new artists/events must be delivered within 1 second

Implemented Solutions

- SignalR for real-time communication in chatrooms and notification system

- Entity Framework Core with SQL Local DB for optimized database operations

- Lazy loading for efficient data retrieval

- MVC architecture for logic separation and efficient view rendering

2. Reliability

Requirements

- Created playlists must maintain 100% data consistency

- Friend connections and event history must be accurately reflected at all times

- System features must work consistently across different browsers and devices

- Music matching system must maintain 99% accuracy in friend suggestions

Implemented Solutions

- SQL Local DB ensuring ACID compliance

- Entity Framework Core for relationship management and data integrity

- Repository pattern implementation for consistent data access

- Client and server-side validations for data consistency

3. Availability

Requirements

- Application must maintain 99% uptime

- Chatrooms must be instantly available after creation of new events/genres/artists

- Playlist access must remain uninterrupted even during high traffic

- Real-time features (chat, notifications) must maintain 95% availability

Implemented Solutions

- SignalR with automatic reconnection for maintaining real-time connections

- ASP.NET Core's built-in middleware for efficient request handling

- Connection resiliency through Entity Framework Core

- Efficient database connection management through connection pooling

4. Security

Requirements

- Private playlists must be accessible only to authorized users

- Personal data must be protected with industry-standard encryption

- Chatrooms must be protected against spam and abuse

- User authentication must prevent unauthorized access attempts

Implemented Solutions

- ASP.NET Core Identity for authentication and authorization

- Customized Identity views to match application requirements

- Authorization attributes for endpoint security

- Protection against SQL injection through Entity Framework Core's automatic parameterization of LINQ queries

5. Usability

Requirements

- Users must be able to create playlists in 3 steps or less

- Friend finding system must require no more than 2 clicks

- Event history must be filterable within 1 click

- Chatroom participation must be achievable within 2 clicks

- New user onboarding must take less than 5 minutes

Implemented Solutions

- MVC design pattern for clear responsibility separation

- Client-side validation for instant feedback

- Real-time notifications through SignalR for instant updates

- Custom ViewComponents for reusable UI elements

6. Maintainability & Resilience

Requirements

- Application must maintain core functionality during partial system failures

- Updates must not affect existing user data and connections

- System must recover from failures within 30 seconds

- 99.9% of user experience must remain consistent after updates

Implemented Solutions

- MVC architecture for clear separation of concerns

- Entity Framework Core for database operations

    o Automated CRUD operations through DbContext

    o Management of complex relationships between users, playlists, and events

    o Simplified data access through LINQ queries

- Dependency Injection for reduced component coupling

- Global error handling through middleware

- Logging through ASP.NET Core built-in logging providers