Non-Functional Requirements (NFRs) and Chosen Architectural Solutions for MusicMatch

Overview

Non-functional requirements refer to constraints or requirements that dictate how the system performs certain functions, as opposed to what functions the system performs. They are crucial for ensuring the quality, reliability, and usability of the system.

1. Performance and Scalability

Requirement: The system must handle a large number of concurrent users with minimal latency.

Architectural Solution: The system uses optimized SQL queries to minimize response times to less than 2 seconds (tested on a decently large database). Additionally, an optimized backend API and a scalable database service with data replication and partitioning is implemented. Application-level caching reduces access time for frequently used data.

2. Reliability

Requirement: The system must be reliable, functioning correctly under normal operating conditions and capable of handling transient failures gracefully.

Architectural Solution: Fault tolerance will be built into key components of the system. For example, retry mechanisms and timeouts will be implemented for all external API calls and data requests. The system will also incorporate comprehensive logging and monitoring to detect failures and take corrective action quickly.

3. Data Integrity and Redundancy

Requirement: The system must ensure the integrity of all stored data and provide mechanisms to recover data in case of failure or corruption.

Architectural Solution: Database redundancy is achieved through replication and clustering, ensuring data is consistently available. Regular automated backups are taken to prevent data loss. In the event of data corruption or loss, recovery processes will restore the system to the most recent valid state.

4. Security

Requirement: The system must protect against unauthorized access and ensure the privacy of sensitive data.

Architectural Solution: The system includes identity management using .NET Identity, ensuring strong authentication and authorization mechanisms. Input validation is applied to prevent malicious input, and all data is transmitted securely using HTTPS. For a more detailed explanation of our security measures, refer to the **Security Analysis Document**, which provides a comprehensive overview of the security protocols, threat models, and mitigation strategies in place for the system.

5. Availability

Requirement: The system must minimize downtime and ensure continuous operation.

Architectural Solution: Real-time monitoring is tracking the system health, and automated backups are taken regularly to ensure quick recovery in the event of any issues. Redundant components ensure system availability during downtimes.

6. Usability

Requirement: The system must be user-friendly, ensuring a positive experience for all users regardless of technical expertise.

Architectural Solution: The user interface (UI) is designed with intuitive navigation and accessibility in mind, adhering to established UX/UI best practices. The system undergoes regular user testing to ensure ease of use, and interactive guides are provided to help users get started quickly. User feedback is collected regularly to improve the system's design and features. Documentation and support materials are provided to assist users.

7. **Portability**

Requirement: The system must be compatible across different devices and screen sizes, ensuring users can access it from multiple platforms.

Architectural Solution: The system will be built with a responsive design, ensuring it works seamlessly on a wide range of devices, including desktops, tablets, and smartphones. Cross-device testing will be performed to guarantee functionality across various platforms and browsers.

8. **Maintainability**

Requirement: The system must be easy to maintain, with clear documentation and code that is easy to understand and extend.

Architectural Solution: The codebase will follow industry best practices, using clean code principles and design patterns to ensure readability and maintainability. Automated unit and integration tests will be used to ensure functionality and prevent regression. The system will have version control (e.g., Git) in place to manage changes, and all code will be properly documented. A clear process for releasing updates and patches will be defined to maintain the system's health.

Conclusion

These non-functional requirements ensure that the system meets high-standards in terms of performance, availability, security and user experience. By implementing these architectural solutions, we can provide a robust, reliable, and maintainable system that satisfies both the functional needs of the users and the quality standards demanded by stakeholders.