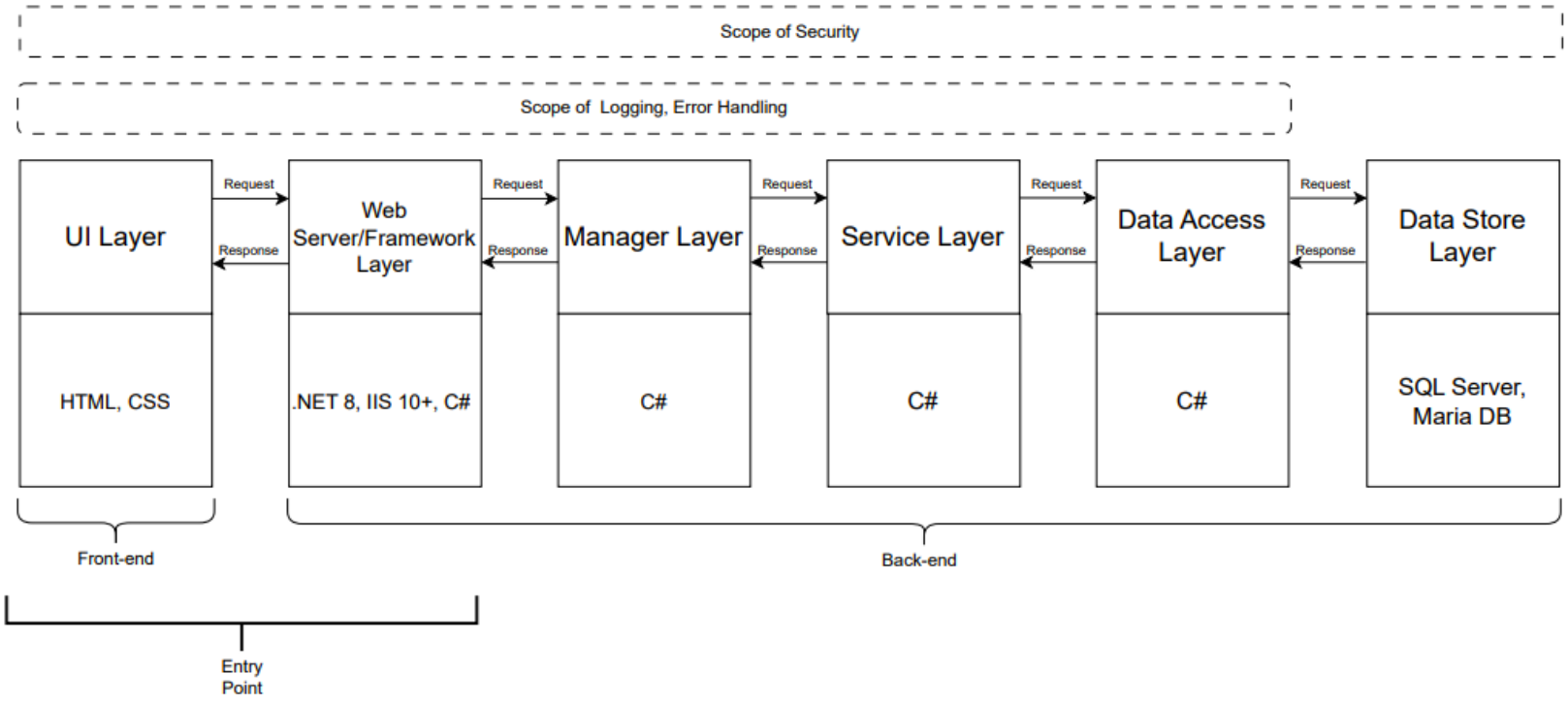
High Level Design Document

Version History

Version	Date	Authors	Changes
1.0	11/22/2023	Jacob, Nathan	Document Initialization
1.1	11/22/2023	Jacob	Added Rough Diagram, setup layers
1.2	11/22/2023	Jacob, Nathan	Added descriptions for each layer and abstractions for core components


Scope of Security
Scope of Logging, Error Handling
UI Layer
HTML, CSS
Request
Response
Web Server/Framework Layer
.NET 8, IIS 10+, C#
Manager Layer
C#
Service Layer
C#
Data Access Layer
C#
Data Store Layer
SQL Server, Maria DB
Front-end
Back-end
Entry Point

Layered Architecture:

User Interface Layer

The User Interface section of our layered architecture represents the first entry point in our front end architecture, where users interact directly. The Musi-Cali app will be leveraging ECMAScript(ES6+), to ensure dynamic and responsive interactions for the users. HTML will be our backbone, which will allow an accessible manner of structuring content. For CSS, it will bring style and aesthetics to the Musi-Cali track editor and any interface.

As for the design of the diagram and the placement of the UI layer, there is a bidirectional flow between the UI and the web framework layer. Form submissions and clicks within the Musi-Cali platform will be processed through the UI layer for input validation on the client-side. These are then sent to the next layer of architecture, the web framework layer.

Web Server/Framework Layer

The Web Framework layer serves to be the intermediary between UI and our underlying business logic, where it handles incoming requests from UI. This leverages ECMAScript with ES6+ standards, where our web framework is to deliver a responsive experience. This layer along with UI, will serve to be our presentation layer being complemented by HTML and CSS. IIS10+ will also be used, which will orchestrate the deployment and hosting of musi-cali. .NET 8 will be our framework used for backend web architecture.

As for the flow of the user requests that pass through the web server, the next layer, the manager layer, invokes the business logic and data operations for specific features to musi-cali. The web server layer also receives refined outputs from the manager layer to translate into appropriate HTTP responses.

Manager Layer

At the Manager layer, our app leverages C# for encapsulating intricate business logic for executing algorithms to create the Musi-Cali digital workstation and the many features it comes with. Implementing TypeScript makes it easier to maintain the Musi-cali application’s logic.

From the web framework, comes user requests, data payload that necessitate specific business logic executions. Within the manager layer, these requests are validated and processed, guiding execution for the Musi-Cali apps needs.

Service Layer

In our architecture, the service layer acts as a pivotal bridge between the manager and data access layer, optimizing the business logic enforcement and ensuring it is applied through many tiers. The service layer utilizes C# 10+. Business rules, validations, and computations specific to the musi - cali app's domain are sent to this layer from the manager layer. The outputs are related to data access, where they are used for persistent storage and retrieval.

Data Access Layer

The data access layer of our layered architecture is going to be the link between the data store layer and the service layer. This is going to be a C# 10 program written to interact with the data store to retrieve data and return it to the service layer to be distributed to the functionality requesting the data.

The data access layer is where all of the code to manipulate and interact with the data store will be contained such as the connection details and SQL queries. This will be the only direct connection to the data store so that there aren't any data leaks.

Data Store Layer

The data store section of our layered architecture is going to be the centerpiece of our back-end where all the data from the app will be stored. For Musi-Cali, the data store will be a SQL database engine so that we have full control over the design.

The data store layer is placed at the deepest layer of the diagram with a bidirectional flow so that it is secure and the system has limited access to it. In order to access data within the data store, the data access layer needs to make a request of the data store layer which will then return the requested data, if available, to the data access layer. This is reasoning for the bidirectional flow in the diagram.

Abstraction and Flow of Control:

- Error Handling:
    - An error is defined as anything not regarded as a successful outcome or intended usage for a functionality. Should an error occur, a system message will display

which tells the user what went wrong as well as logging the error for system troubleshooting.

- Logging:
    - All operations will be logged in order for our system to be transparent enough to debug the application. This is especially true for error and warnings so that we can more easily diagnose bugs.
- Data access:
    - The database will be accessible to all users, administrators, and automated services. User type (Artist or Administrator) will determine what parts of the database can be accessed and changed
- Security (Authentication & Authorization):
    - Authentication:
        - Authentication will be handled using a username and One-Time-Password (OTP) to login.
        - After authentication, the user will either be assigned an artist role or administrator role depending on their account.
    - Authorization:
        - Authorization will take place anytime a user tries to access a protected resource.
        - User is either allowed or not allowed to access the protected resource depending on their user type and permissions.
- Flow of Control:
    - The flow of control for the core components would be:
        - security to make sure the user has access to a particular protected resource
        - Data access within the protected resource
        - During an operation attempted within the protected resource, if an error occurs:
            - First the system displays a message to tell the user what went wrong
            - Then the system logs the error for system transparency and system diagnostics