**Question 1**

1. Data Structures

* I will implement a standard Binary Search Tree data structure, the nodes will store an Audio object. I could use the C# SortedSet<T> which is a self-balancing binary tree.
* A Queue will be used as a song line-up feature,
* And a Stack to use as the song history which will be accessed by the previous song button.

2. Hashing Techniques

* I will be storing user hashes and salts along with usernames in the Server application memory, and this user list will be serialized when Server is closed.
* When the user has logged into the program (using Client-Server NamedPipe connection) they will be granted access to the Music Player application depending on their security level (e.g. admin and regular user can access, while a guest cannot)
* The users table will contain user login records. Plain-text passwords will not be stored.
  + When opening the application, the user will be prompted to log in or create an account. This is where hash validation will be used.

3. Sorting Algorithms

* I will use merge sort algorithm to sort the playlist ListBox by name, duration in ascending or descending order. The merge sort will not sort the binary tree (which is already naturally ordered), but will sort a corresponding List<> object.
* The List<T> object will be used as a ListBox dataSource, or the current playlist, so the user can sort the List by Audio properties, ascending and descending. This will not sort the underlying binary tree.

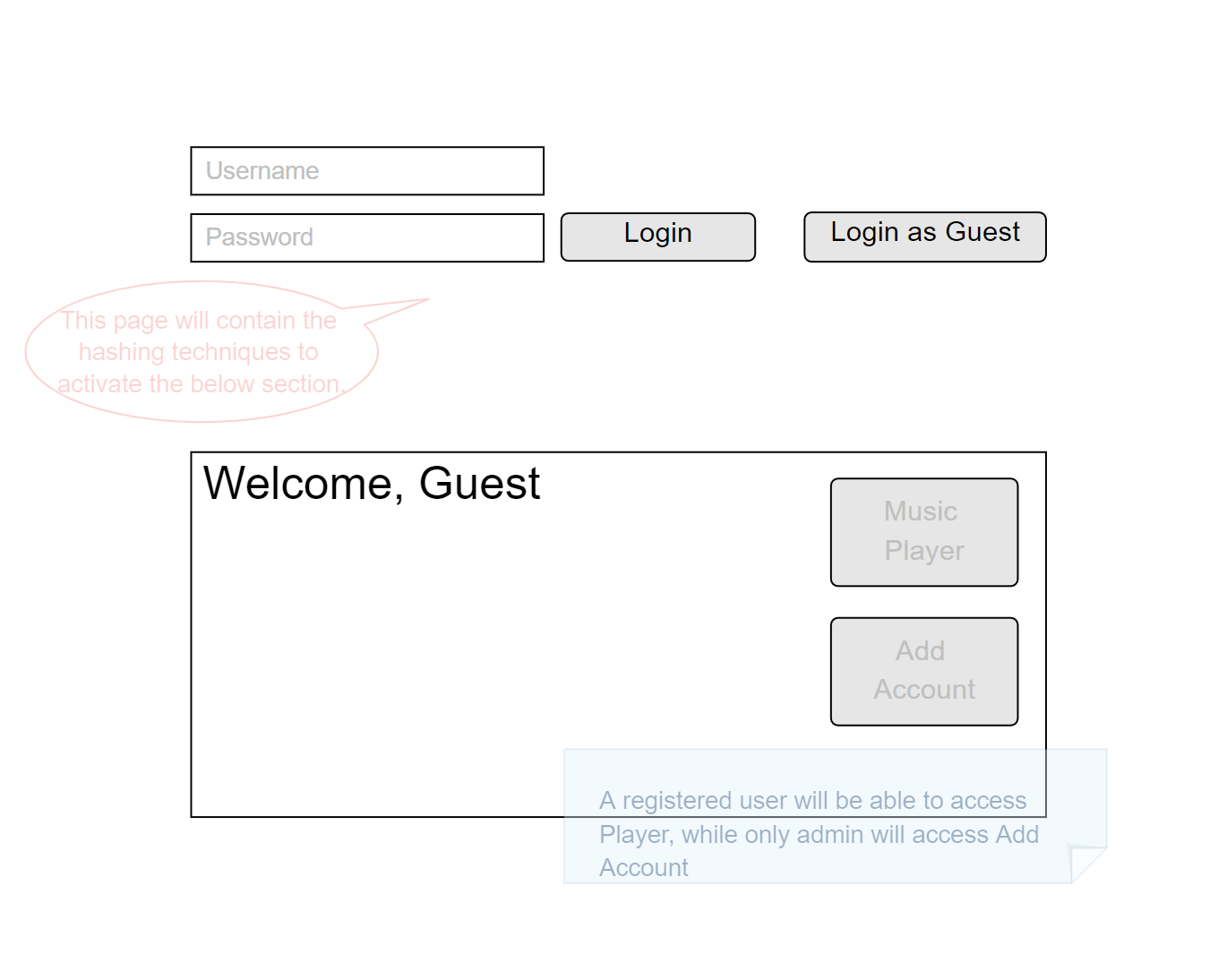
4. I will be using a Binary Search algorithm, for searching within the current playlist or throughout the whole library. The search criteria will require the search string to be more than 4 characters.

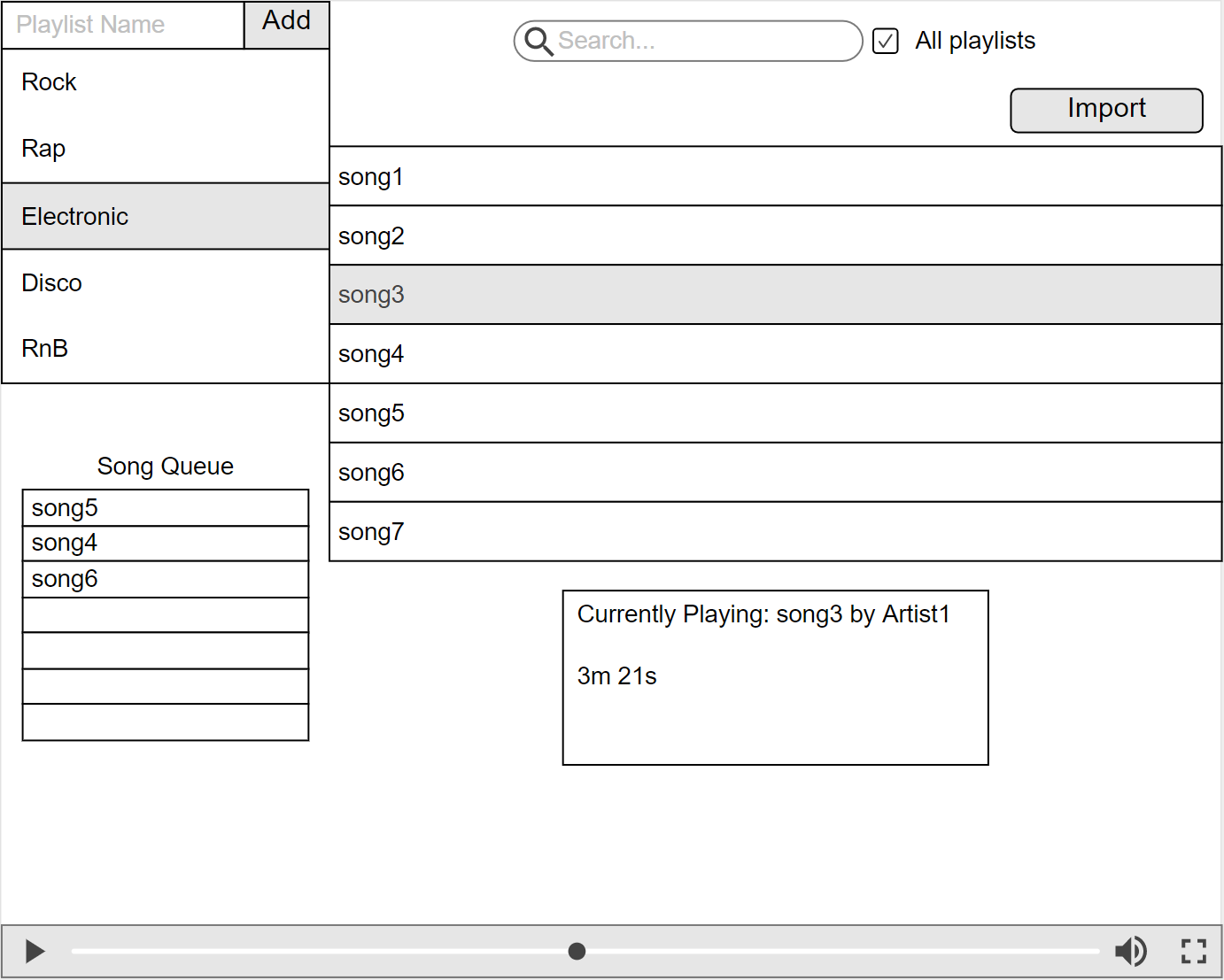
5. I will use CSVHelper to output a CSV file of all playlists, and a file containing all music.

* PlaylistID, Name, Description, TotalDuration …
* AudioID, Title, Duration, Path …

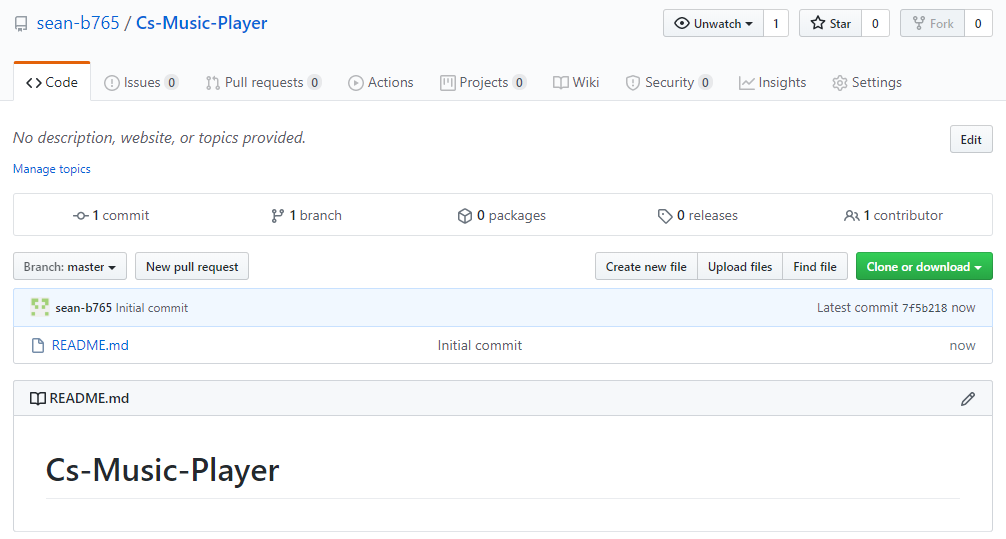
6. CSVHelper documentation can be found at <https://joshclose.github.io/CsvHelper/api>

7. GUI mockup:





8. I will be using GitHub and will push changes when a stable feature is newly available.



9. Coding standards I will enforce are:

* C# naming conventions, such as PascalCase for methods and classes, but camelCase for local variables or method parameters.
* Code should be sufficiently commented, but unwanted comments should be avoided.
* Proper indentation
* Code should not repeated; instead use methods.
* Code should be written with portability in mind; avoid hard-coded values which may change between environments.

10. Unit Tests: After a module is implemented, I will perform unit tests so that errors can be picked up early.

System Test: will be performed on the whole application, testing the overall functionality, and trying to produce any errors.