{TypeMania}

Preliminary Project Website:TypeMania.github.io

CSCI 441 – Project Report 2 Completed

Group 2:

Shaka Elmore

Nargis Sultani

Heather Vroman

Alexandra Stevens

Oct 16, 2022

Table of Contents

Individual Contributions

	Shaka Elmore	Nargis Sultani	Heather Vroman	Alexandra Stevens
Discord Management	85%	5%	5%	5%
Github Management	25%	25%	25 %	25%
Member Contribution	25%	25%	25%	25%
Meeting Minutes	70%	0%	30%	0%
Deployment Platform Research	0%	100%	0%	0%
Database Research	0%	50%	0%	50%
Google Doc Creation	25%	25%	25%	25%
Framework Research	0%	0%	0%	100%
Main Menu Design	5%	5%	85%	5%
System Operation Contracts	0%	0%	100%	0%
Conceptual Model	0%	0%	0%	100%
Data Model and Persistent Storage	0%	100%	0%	0%
Interaction Diagrams	0%	0%	100%	0%
Class Diagram	0%	100%	0%	0%
Data Types/Operation Signatures	0%	0%	0%	100%
Traceability Matrix	100%	0%	0%	0%
Project Management	75%	25%	0%	0%
React Template Code (Github)	0%	0%	0%	100%
React Nav Skeleton/Login Form	0%	0%	0%	100%
Algorithms & Data Structures	100%	0%	0%	0%
Look into concurrency requirement	0%	100%	0%	0%

Design of Tests	0%	0%	100%	0%
Developer Guide & Tutorial Videos	0%	0%	0%	100%
Github Branch Maintenance	20%	20%	20%	40%
UI Design & Implementation	0%	0%	0%	100%
Rest API(MVC)/Backend	0%	0%	0%	100%
mongoDB creation	0%	0%	0%	100%
Heroku account creation	0%	0%	0%	100%
Frontend Form Integration to API	0%	0%	0%	100%
Game Scene Draft	40%	0%	40%	20%
Editing Report Two	20%	40%	20%	20%

1. Analysis and Domain Modeling

1.1 Conceptual Model

1.1.1 Concept definitions

The domain model concepts and corresponding responsibilities were derived from the thirteen previously defined system use cases defined in Report 1. Based on these use cases, the domain model is made up of the interface, game, controller, validator, and database concepts. See Figure 1 below illustrating the full domain model diagram. The detailed domain model concepts and responsibilities are summarized in Table 1.

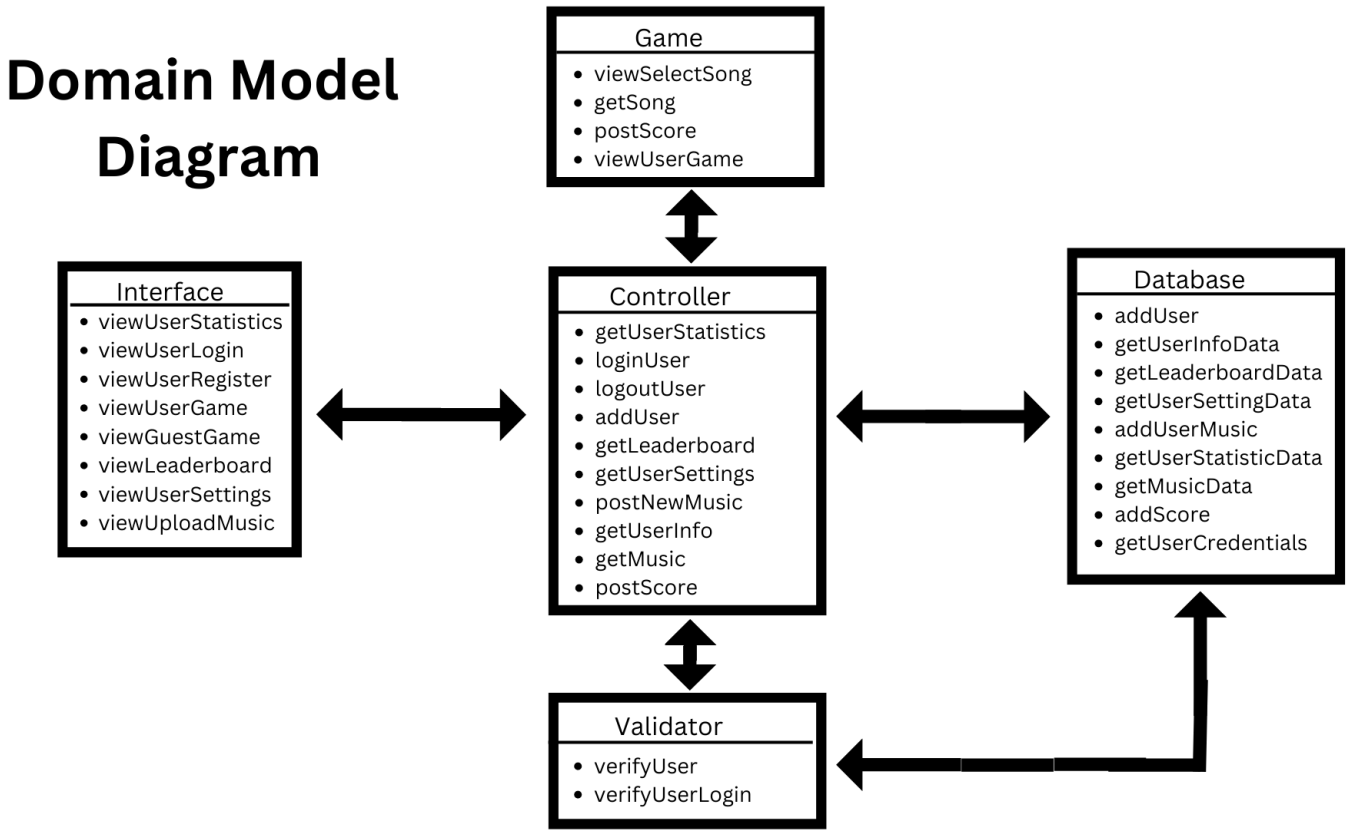

Figure 1: The system's domain model diagram
https://www.canva.com/design/DAFNxvymw8Q/W2q7h9VMPLQ-_M67ITduGA/edit?utm_content=DAFNxvymw8Q&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Responsibility Description	Type	Concept Name
Rs1. React Website that features user statistics, user login, user registration, user game, guest game, leaderboard, user settings, and user music upload screens; UI with game component with database connection and get and post communication	K	Interface
Rs2. Coordinate the actions of all concepts, including the interface, game, validator, and database; coordinates all get and post actions to and from the database; provides data from database to create a custom user experience, game play scores, and leaderboards	K, D	Controller
Rs3. Validates posted user credentials with stored user credentials in the database; verifies new user is not a duplicate user in the database	K	Validator
Rs4. Contains all game functionality; views generic guest game UI or registered user game UI through getUserInfo in controller; gets song library and gets user chosen song for gameplay; views player score during game play, posts user scores to database for statistics and leaderboard	K, D	Game
Rs5. Stores all user data and default song data and completes get and post requests for data; requests come through validator and controller from interface and game concepts	K, D	Database

Table 1: Domain model concepts and responsibilities

1.1.2 Associate definitions

In order to achieve the defined responsibilities shown in Table 1, all of the listed domain model concepts have association with other concepts throughout the model. These concept association descriptions are listed below in Table 2.

Concept Pair	Association Description	Association Name
Interface«Controller	Interface makes get requests to Controller, Controller processes request communicating with the database and sends back data to Interface to display pages; Interface makes post requests to Controller and Controller processes the posts requests	Generates request, Conveys result,
Game«Controller	Game makes get requests to Controller, Controller processes request and sends back data to Game display on game interface; Game makes posts to Controller and Controller processes the posts requests	Generates request, Conveys result
Controller«Validator	Controller makes requests to Validator to verify user credentials, Validator processes request by communicating with database and sends back data to Controller to move to next step of current process	Generates request, Conveys result
Validator«Database	Validator makes requests to Database to verify user credentials, Database processes request and sends back data to Validator to return a true or false verification	Generates request, Prepares request, Conveys result
Controller«Database	Controller makes get and post requests to Database, Database processes request; Controller then moves to next step in current process	Generates request, Prepares request

Table 2: Domain model concepts associations

1.1.3 Attribute definitions

When examining the above listed concepts and their relationships, we must also detail the various attributes that accompany each concept. The attributes have various functions that contribute to the overall actions or goals of the concepts. The system's concept attributes are described below in Table 3.

Concept	Attribute	Attribute Description
Interface	viewUserStatistics	Displays user statistics page and data of logged in registered user; sends request to Controller via getUserStatistics
	viewUserLogin	Displays user login form; accepts inputs; sends request to Controller viewUserLogin
	viewUserRegister	Displays user register form; accepts inputs and sends to Controller addUser
	viewUserGame	Displays home game page with user data; sends request to Controller to getUserInfo
	viewGuestGame	Displays default home game page
	viewLeaderboard	Displays updated leaderboard on home game page; sends request to Controller to getLeaderboard
	viewUserSettings	Displays registered user settings; sends request to Controller to getUserSettings
	viewUploadMusic	Displays upload music form; sends request to Controller to postNewMusic
Controller	getUserStatistics	Requests user statistics from database and returns results
	loginUser	Sends request to Validator to verifyUserLogin; returns result; returns login cookie key or error message
	logoutUser	Clears login cookie

	addUser	Requests user statistics from data base and returns results
	getLeaderboard	Requests leaderboard data from database and returns results
	getUserSettings	Requests user setting data from database and returns result
	postNewMusic	Sends post request to database to store new user uploaded song to user song database
	getUserInfo	Requests user info data from database and returns result
	getMusic	Requests song data from database and returns result including song and songmap
	postScore	Sends post request to database to store new user score
Validator	verifyUser	Requests if user credentials already exists; processes request and returns result
	verifyUserLogin	Requests if user credentials are correct from database; processes and returns results
Database	addUser	Processes addUser and adds user to database
	getUserInfoData	Processes getUserInfoData and returns info data matching requested user
	getLeaderboardData	Processes getLeaderboardData and returns leaderboard data
	getUserSettingData	Processes getUserSettingData and return setting data matching requested user
	addUserMusic	Stores song in user music database
	getUserStatisticData	Processes getUserStatisticData and return statistic data matching requested user
	getMusicData	Processes getMusicData and return info data matching requested song

	addScore	Stores score data for matching user
	getUserCredentials	Returns the data for requested user login info
Game	viewSelectSong	Displays select song game menu; sends request to get song data to populate dropdown to Controller to getMusicData
	getSong	Requests song data and songmap to Controller to getMusicData
	postScore	Sends post request to Controller to addAscore
	viewUserGame	Displays user game interface; sends request to get user info to populate interface to Controller to getUserInfoData

Table 3: Domain model attribute definitions

1.1.4 Traceability matrix

Below is Table 4, which illustrates the traceability matrix for the described domain model. This table shows the relationship between the various domain model concepts and the system use cases described in report 1.

Domain Model	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13
Interface	X	X	X		X	X	X	X	X	X	X	X	X
Controller	X	X	X	X	X	X	X	X	X	X	X	X	X
Validator		X											X
Database		X	X	X	X	X	X	X	X	X	X	X	X
Game	X		X	X	X	X	X	X					
Total Priority	3	4	4	3	4	4	4	4	3	3	3	3	4

Table 4: Domain model traceability matrix

1.2 System Operation Contracts

Register	
Precondition	The user cannot be an existing user. This will be determined via the information the user enters. If there is any information which is already in the system, the user will be prompted to login with a message communicating to the user why they cannot create an account with said information.
Postcondition	After the user creates an account their personal information is stored in the database and any future account cannot be created via the same information. With a registered account additional options are available to the user now that the system adjusts compared to a guest. This includes tracking typing speed and keeping a record of all user history which can be viewed in statistics.
Changes made to the system	Changes made to the system are the users information stored in the database, and registered user only prompts which become available to the user, and the internal database recording the users history which can be viewed at a later time if desired.
Login	
Precondition	To login the user must already be a registered user. This information will be determined via login information ensuring the username has a valid account, and the password matches said users password. If the password is incorrect but the username is correct the user can enter their password again or go through the steps to reset their password.
Postcondition	Login and access the system as a registered user. This brings the user all the features as a guest user plus features that can be only available after successfully logging in such as their full user history, personal user

	information, and personal statistics.
Changes made to the system	The changes made to the system are ensuring the information the user inputs matches said information in the database. After successfully logging in the system outputs user only features such as full user history and personal information.
Play Game	
Precondition	There is no precondition to play the game aside from landing on the webpage.
Postcondition	After successful gameplay the user can view their score compared to other users.
Changes made to the system	There are no changes made to the system before gameplay. After game play the score regardless of user status is stored in the database so it can be used to compare to other users. If the user is a registered user their score goes into their personal database to be used to track their progress made to their typing efficiency progress. This data can be applied to graphs, top scoring charts, and user history.
Statistics	
Precondition	To view the statistics the user must be logged in, and land on the correct section of the webpage to analyze their personal statistics.
Postcondition	After the user lands on the correct section of TypeMania, and is successfully logged in they can view their statics.
Changes made to the system	The only change made to the system is in the logged in users history. The database stores that said users viewed their statistics.

Table 5: System Operation Contracts

1.3 Data Model and Persistent Data Storage

PostgreSQL is used to store the data. There are three tables:

- Users
- Songs
- Statistics
- Missed Characters

Below is the ER Model for the database. The Users table will store the user info, credentials, and a unique ID to reference to a specific user. The Statistics table will store the statistics for a specific game and user. The Songs table will store a list of songs so the users could select from. Both the Users and Songs tables have a one-to-many relationship with the Statistics tables. The Missed Characters table stores the missed characters and its count for a specific Game. It has a many-to-one relationship with the Statistics table.

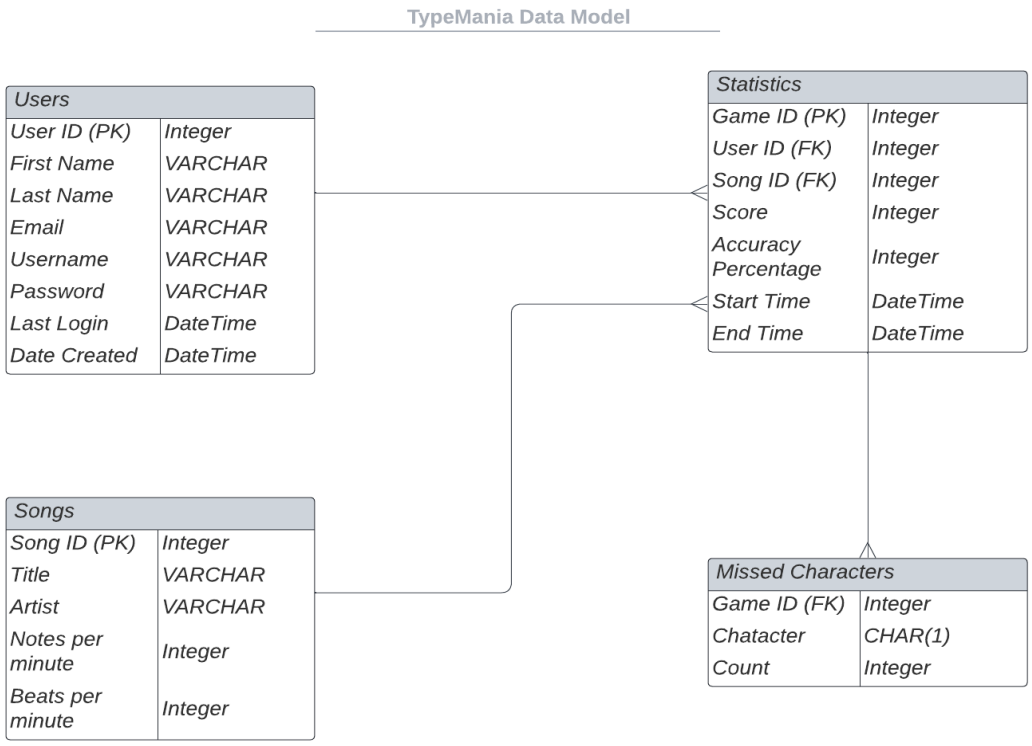

Figure 2: The system's ER Diagram

2. Interaction Diagrams

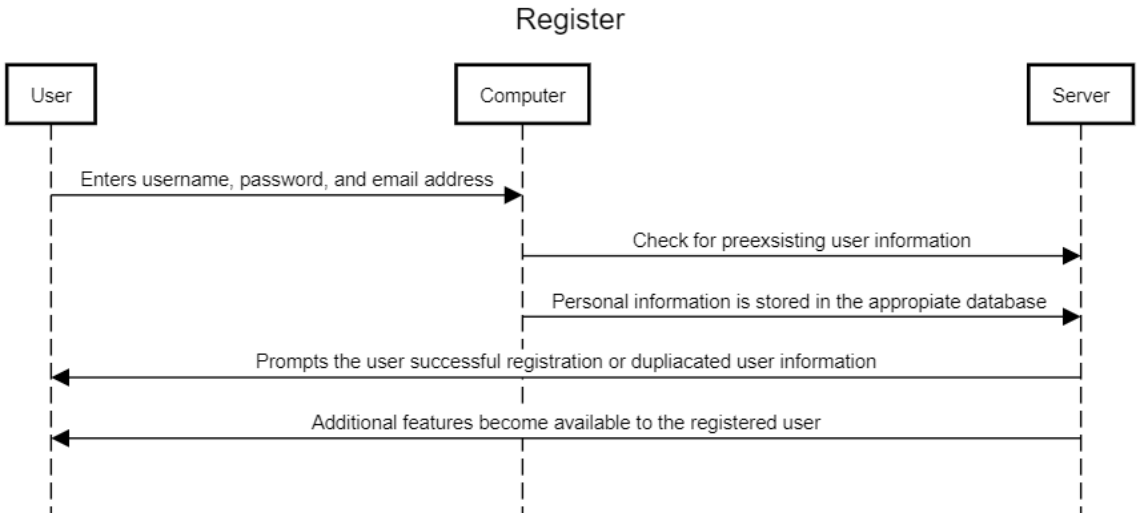

Figure 3: Register Interaction

Reducing coupling is used when the user is registering as it could be on the same page as login/ logout. This makest registering as a user independent, making the code much easier to read. Design for flexibility is applied when the user is registering because it allows additional features for only users to be displayed after the user logins compared to set features. Also, the user can change their personal information which is stored in the database as long as said information does not match preexsisting information being stored, which srves as another flexible option.

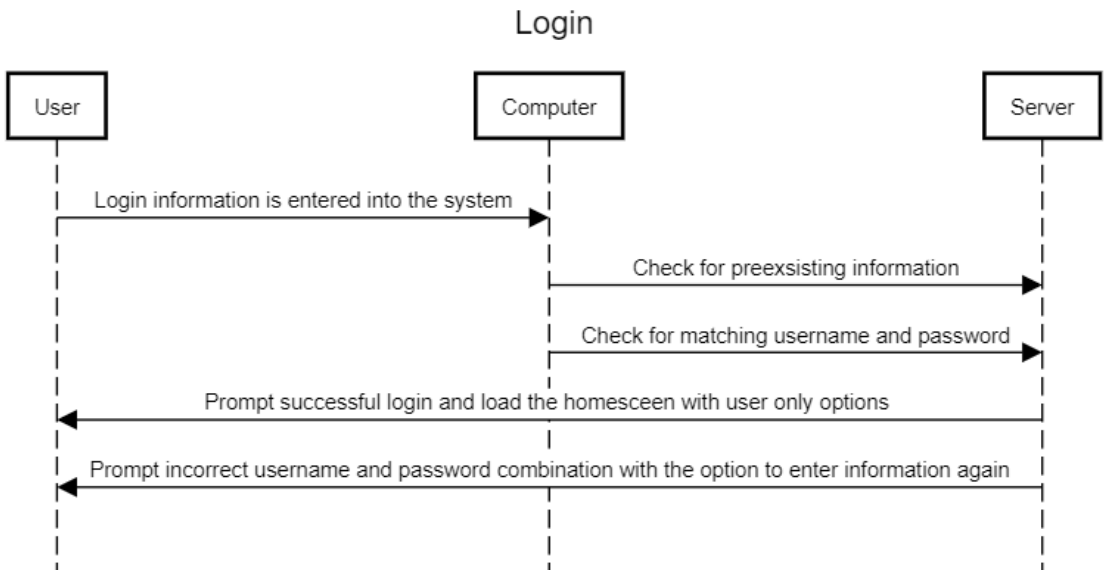

Figure 4: Login Interaction

Reducing coupling is used when the user logging in or out as it could be on the same page as registering as a user. This makes loggin in or out more independent making the code much easier to read. Increased reusabiltiy is applied making it easier to login if the user inputs the wrong information due to a typing error or other mistake. Instead of having to find the login link again, the application will prompt the user that there has been an error with the information entered and bring the screen to try loggin in again to the users screen.

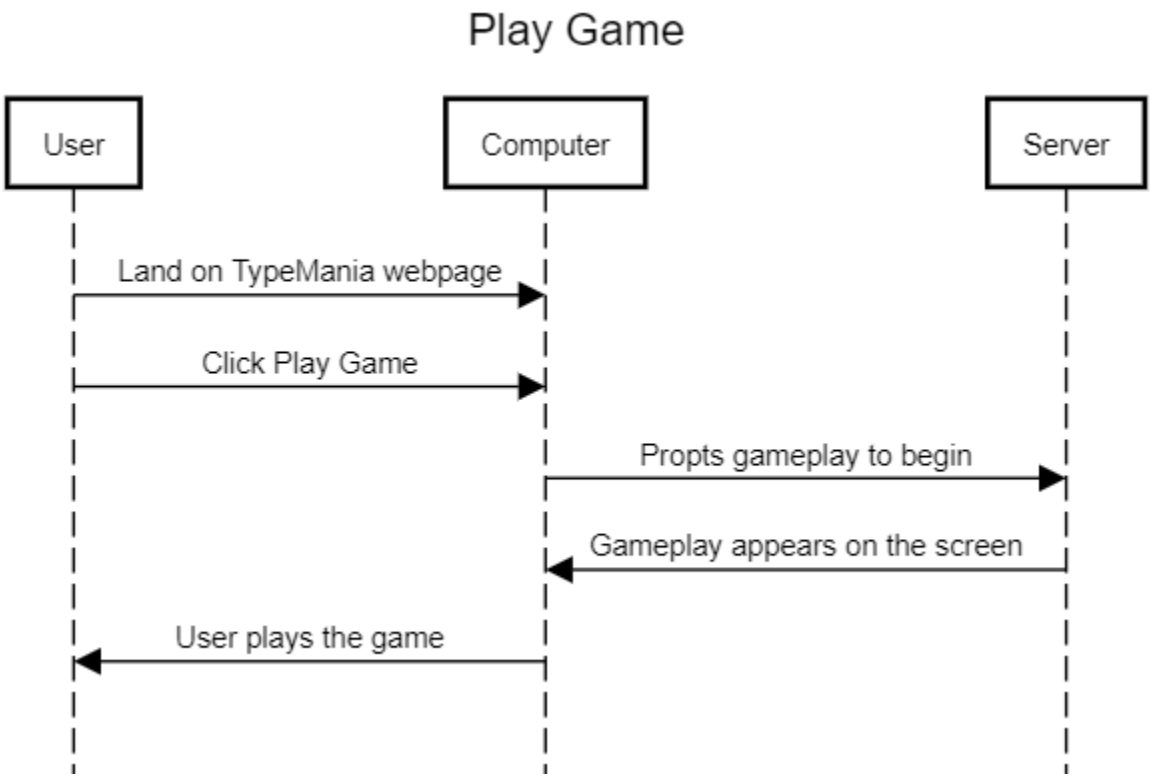

Figure 5: Play Game Interaction

Design for flexibility is applied to the gameplay because as the basic prompts will remain the same, the functions within are subject to change as time progresses. Abstraction will also be applied to game play. This allows the code to be duplicated for gameplay purposes rather than writing new code for every scenario the user may encounter.

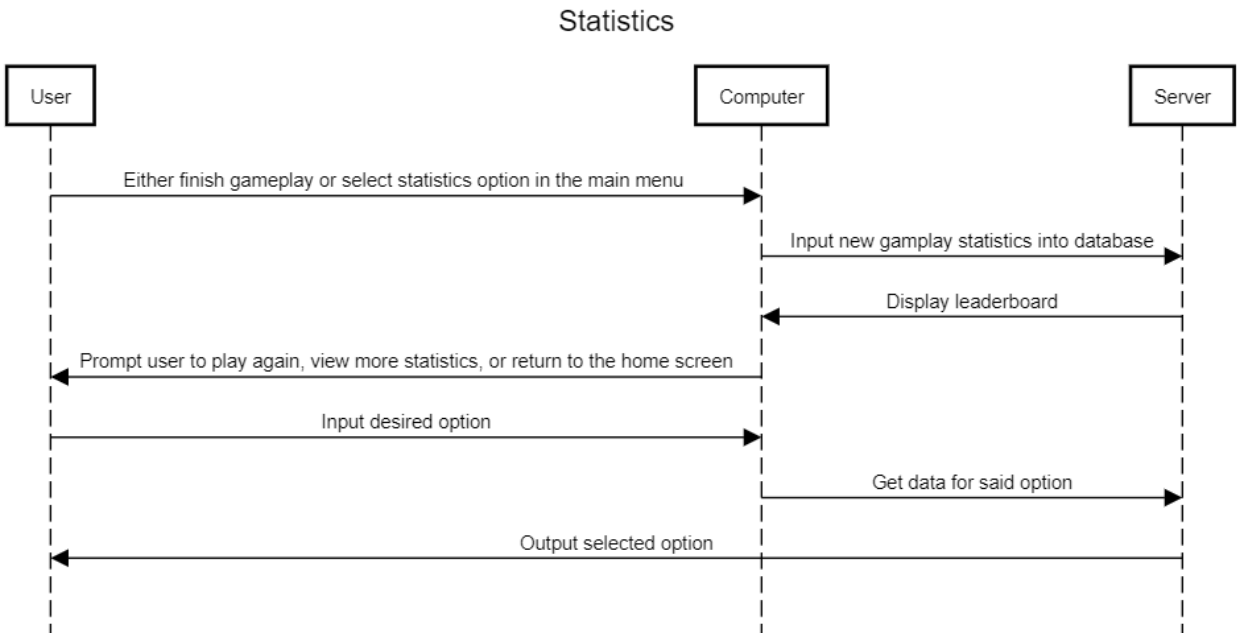

Figure 6: Statistics Interaction

Due to the complexity of the statistics page the divide and conquer design principle is applied. The larger system is divided into smaller systems while the diagram shows the user landing on the leaderboard after game play the smaller systems the user can select. These consist of play again which leads to gameplay, returning to the homescreen, or viewing more statistics. Viewing more statistics allows the user to access their full history, past scores, and typing efficiency progress hence the divide and conquer strategy.

3. Class Diagram and Interface Specification

3.1 Class Diagram

As illustrated in the diagram, there will be a total of five classes for this application: User, Song, Game, Statistics, and Leaderboard.
The relationships between the classes are described below:
Each User can play multiple Games at different times, however one Game can be played by only one User at a time. Each User can upload multiple Songs and play only one song at a time. Only one Song can be played during each Game. The Statistics will have multiple Games for each user. The LeaderBoard will have a list of Users and Games, it has a many-to-many relationship between both the User and Game.

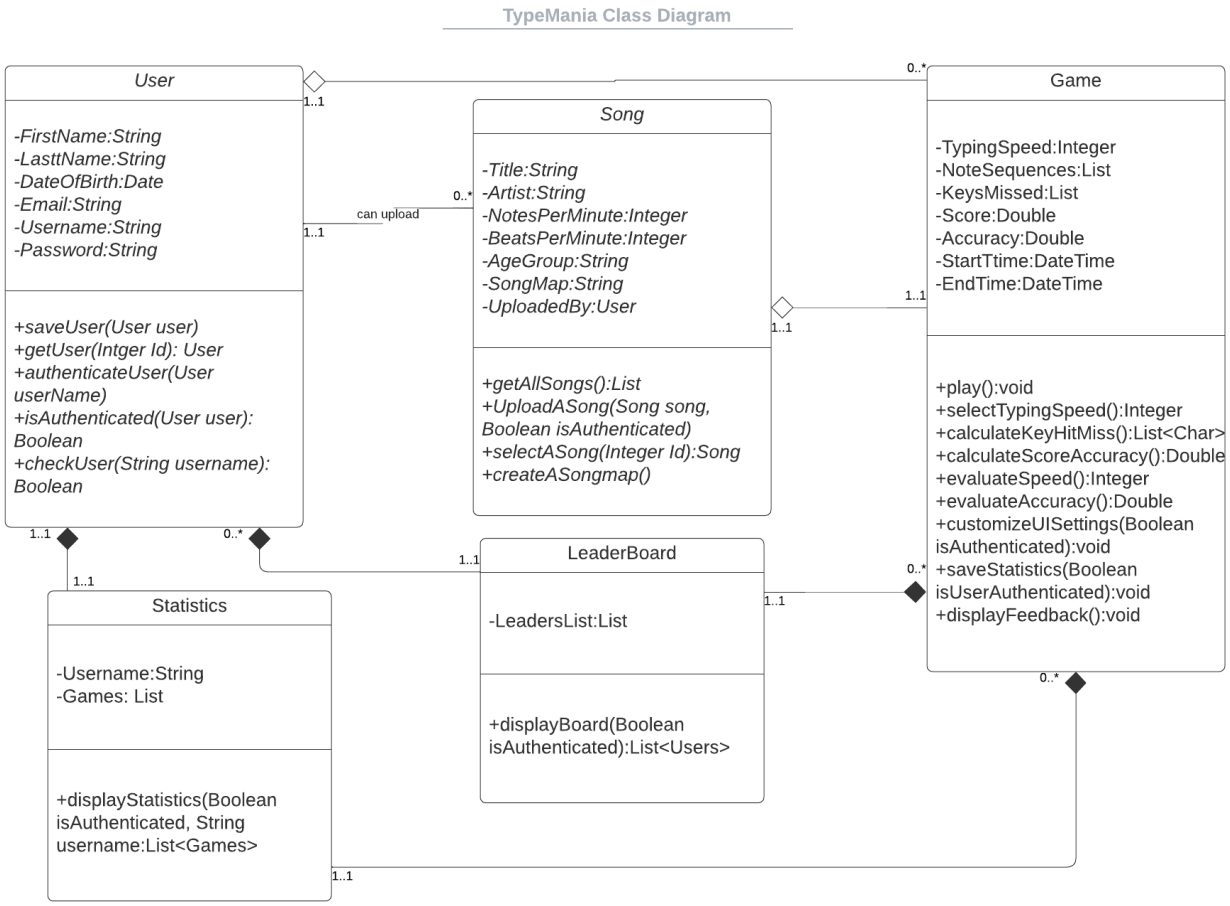

Figure 7: TypeMania Class Diagram
https://lucid.app/lucidchart/ca78cfcf-e53f-4b87-be80-d883e64634bd/edit?viewport_loc=-255%2C61%2C2603%2C1245%2C0_0&invitationId=inv_e56a1c43-0ee5-4824-9e0a-5d2f24c2957e#

3.2 Data Types and Operation Signatures

I. User Class:

User Class	
The user class performs all functions related to user information including saving a new user for registration, getting user information, and authenticating the user.	
Attributes	
DataType	**Description**
-FirstName: String	This stores the user information: first name
-LastName: String	This stores the user information: last name
-DateOfBirth: Date	This stores the user information: date of birth
-Email: String	This stores the user information: email
-Username: String	This stores the user information: username
-Password: String	This stores the user information: password
Operations	
Signature	**Description**
-saveUser(User user)	This adds the user information to the database as a new user
+getUser(Int Id): User	This returns the user information data matching the user id parameter
-authenticateUser(User username User password)	This authenticates the user information based on username and password
-isAuthenticated(User user): Boolean	This returns the boolean value for if the user is authenticated
-checkUser(User username): Boolean	This returns the boolean value for if the username parameter matches an existing user

Table 6. User Class Data Type and Operation Signature Table

II. Statistic Class:

Statistic Class	
The statistic class returns all user statistic data and displays user statistic component to the user interface	
Attributes	
DataType	**Description**
-Username: String	This stores the username data that will display on the statistics page
-Games: List	This stores the game history of the user that will display on the statistics page
Operations	
Signature	**Description**
+displayStatistics(Boolean isAuthenticated, String username):List<Games>	This displays the user statistic data on the user interface for the statistics page

Table 7. Statistic Class Data Type and Operation Signature Table

III. Song Class:

Song Class	
The song class stores all song data and performs all functions to interact with and select the song data	
Attributes	
DataType	**Description**
-Title: String	This stores the song information for the title of a song
-Artist: String	This stores the song information for the artist of a song
-NotesPerMinute: Int	This stores the int value for notes per minute
-BeatsPerMinute: Int	This stores the int value for beats per minute

-AgeGroup: String	This stores the string value for age group of a song
-SongMap: String	This stores the string value for the song map that accompanies a specific song
-UploadedBy: User	This stores the user data of the user that uploaded a specific song
Operations	
Signature	**Description**
+getAllSong():List	This returns the list of all song titles
+uploadASong(Song song, Boolean isAuthenticated)	This performs the action of a user adding new song data to the song database
+selectASong(Int Id): Song	This action selects a specific song and returns the respective songmap
+createASongmap()	This action creates and adds a songmap for the specific song and speed attributes

Table 8. Song Class Data Type and Operation Signature Table

IV. Leaderboard Class:

Leaderboard Class	
The leaderboard class returns leaderboard data and displays the leaderboard data to the user interface for the leaderboard component	
Attributes	
DataType	**Description**
+LeadersList:List	This stores the leaderboard user and score information
Operations	
Signature	**Description**
+displayBoard():List<Users>	This displays the leaderboard data and react component

Table 9. Leaderboard Class Data Type and Operation Signature Table

V. Game Class

Game Class	
The game class stores all game data and game functionality including game user interface as well as game events and actions	
Attributes	
DataType	**Description**
-TypingSpeed: Int	This stores the user user typing speed for game session
-NoteSequences: List	This stores the note sequences for the selected songmap
-KeysMissed: List	This stores the user data for keys missed and key names for a particular game session and song
-Score: Double	This stores the user data for the score of a particular game session and song
-Accuracy: Double	This stores the user data for the accuracy of a particular game session and song
-StartTime: DateTime	This stores the start time of a particular song
-EndTime: DateTime	This stores the end time of a particular song
Operations	
Signature	**Description**
+play():void	This begins the songmap for a game session and start time
+selectTypingSpeed(): Integer	This instantiates the selected typing speed for a game session to build the songmap
+calculateKeyHitMiss(): List<Char>	This performs the calculation and returns the data for key accuracy
+calculateScoreAccuracy(): Double	This performs the calculation and returns the data for game session accuracy

+evaluateSpeed():Integer	This evaluates the user typing speed during the game session
+evaluateAccuracy()Double	This evaluates the user accuracy
+customizeUISettings(Boolean isAuthenticated):void	This returns user game interface settings
+saveStatistics(Boolean isAuthenticated):void	This saves user game statistics to database
+displayFeedback():void	This displays user feedback during game session

Table 10. Game Class Data Type and Operation Signature Table

3.3 Traceability Matrix

The traceability matrix below highlights how our classes relate to our domain concepts. The purpose is to document the motivations behind the creation of such classes. The relationships between classes and concepts tend to be many-to-many. That is, one concept may be connected to multiple classes and one class may be derived from multiple concepts.

Domain Concept	**Derived Classes**	**Description**
Controller	User, Game, Song, Statistics, Leaderboard	The controller receives and manipulates data from throughout the entirety of the system. It is concerned with the coordination of all concepts and their respective classes. For example, the systemization of the Game class requesting a Song object from the database.
Interface	User, Game, Song, Statistics, Leaderboard	The interface must interact with every element of the system to display pertinent information to the user. This differs from the controller in that it doesn't work behind the scenes and instead operates in full view of the user to make them a part of the system.
Validator	User	The validator is purely concerned with authentication during login and registration events. Information entered is compared against or stored into the User class's credentials.
Game	Game, Song, Statistics	The game concept emphasizes gameplay and functionality. On top of the gameplay UI, it also handles the Song Select UI and is responsible for

		posting scores when the user finishes playing a songmap.
Database	User, Game, Song, Statistics, Leaderboard	The database stores data. Classes are inherently a factory for producing organized sets of data so this concept interacts directly with each one of them. Our system will need someplace to hold the objects output by such factories and that is where the database comes in.

Table 11: Traceability matrix linking classes to domain concepts.

4. Algorithms & Data Structures

4.1 Data Structures

Our software must store sequences of integers. The integers are not unique and may be repeated multiple times within the structure. Each integer represents a character that will appear on a note. The notes will appear one after another in the order that the sequence presents them. The initial sequence of integers is linear and requires shuffling. To shuffle effectively, the structure should be able to read/write to an element without having to traverse the entire sequence to reach said element. From what has been stated above we can derive these requirements for our data structure:

- Sequentiality - The structure must support chronology.
- Duplicability - The structure must accept repeated elements.
- Mutability - The structure must allow data within to be modified.
- Lookup - The structure should implement quick access to elements.

Sequentiality is not the strong suit of hash tables and sets void duplicates. Our need for a quick lookup renders linked lists ineffective and mutability is disallowed in tuples. Arrays support chronology, accept duplicate elements, allow data manipulation, and have an effective lookup via indices. Arrays are the necessary blend of flexibility and performance and are our project's data structure of choice for these reasons.

5. User Interface Design and Implementation

Based on the development of TypeMania conducted thus far, there have been some modifications to the design throughout the software's planning and implementation process. More modifications will likely be made as the development of the web app continues. This may be related to design/ease-of-use improvements as well as to improve development and implementation processes.

Firstly, the navigation bar design has been modified and implemented to list a larger variety of options than what was proposed in the first draft. The first draft contained only login and register links in the navigation bar, whereas the currently implemented version contains home, login, statistics, upload song, and register links, see Figure 8. Additionally, the logo is clickable to act as a second homepage navigation link. Having more options in the navigation bar increases the ease-of-use of registered users, specifically related to the statistics and upload song items. With this change, these items are easier to navigate to by having a constant present on the webpage. This strengthens our design's user control and freedom as well as consistency, which are two key components of successful user interface design (Nielson, 2020). Lastly, if a guest user selects a link that requires a registered account, the guest is directed to the login screen, where they can login or navigate to the register page.

Figure 8: TypeMania navigation bar implementation

The login page is very similar to the previously proposed interface design; however, it also contains an additional link to navigate to the registration page. Below the submit button sits the text, "Not registered," followed by a Register link, see Figure 9. This strengthens user control and freedom as well as consistency and help/communication (Nielson, 2020). Moreover, the registration page contains the text "Already registered?" followed by a login link underneath its submit button, see Figure 10. These two additional messages and navigation links also help to improve user ease-of-use, reducing their memory load, following logical communication, and enabling easy, seamless navigation (Nielson, 2020). The similarity between the login and registration form makes it easier for the user to transition between steps. Moreover, each form will display information messages and error messages to guide the user to enter the correct, required input to successfully process the form. This helps to meet successful interface design requirements, including error prevention, diagnosis and recovery (2020).

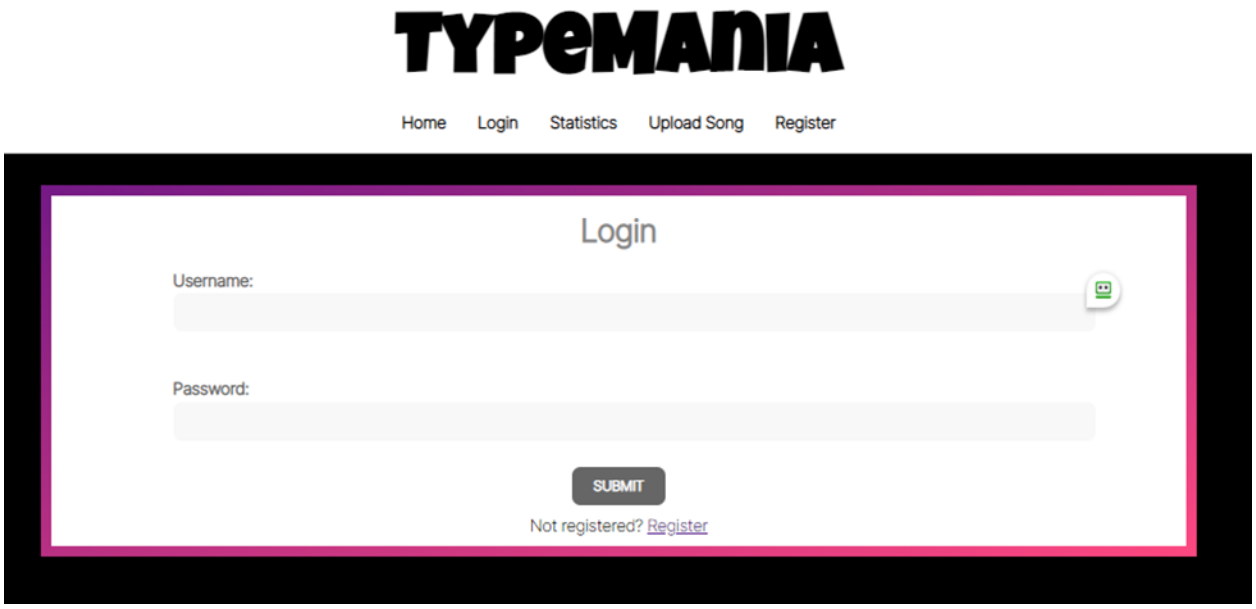

Figure 9: TypeMania login form implementation

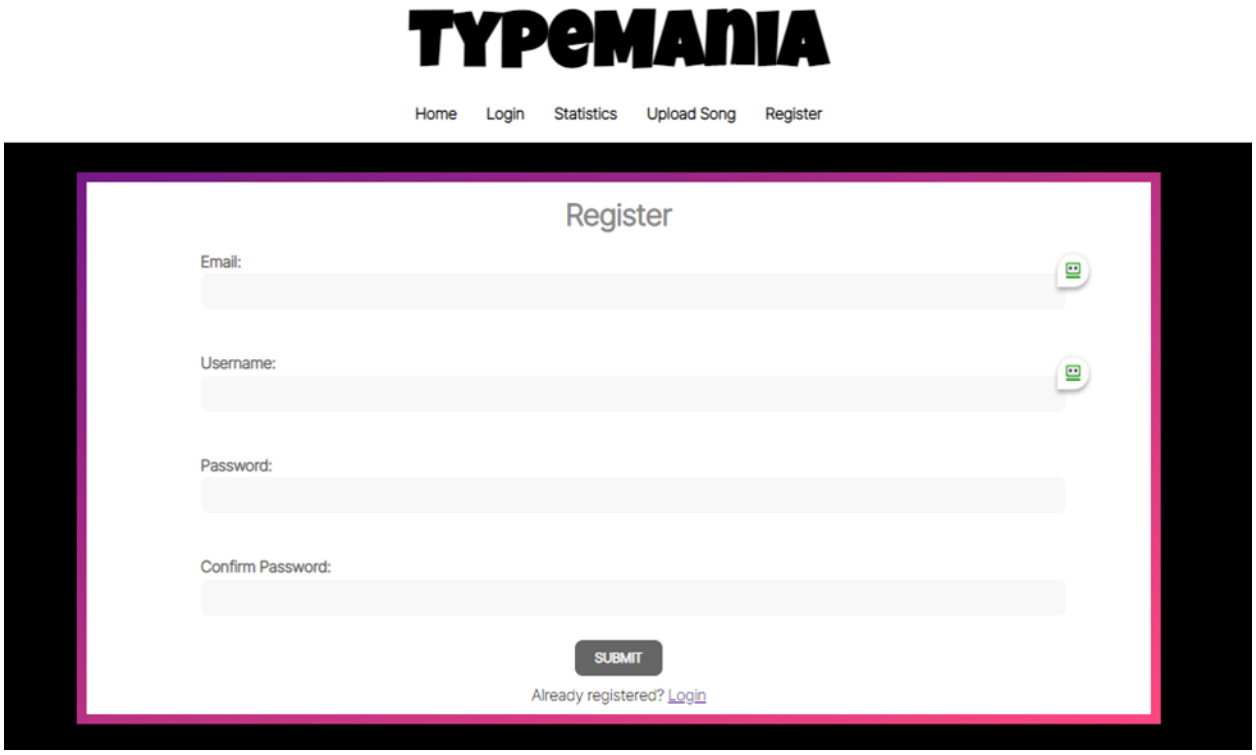

Figure 10: TypeMania registration form implementation

The landing page, see Figure 11, has had some design alterations related to implementation of the various development components and will still be undergoing design improvements. The current in-progress landing page features the navigation bar, leaderboard component, and gaming component. The leaderboard component, magnified in Figure 12, is an in-progress design and will be tweaked and improved in future development. This component helps to promote visibility of system status, which is a key element of good interface design (Nielson, 2020). It also helps to entice users to play TypeMania more and compete with friends and other leaders. The leaderboard component will be visible to both guests and registered users.

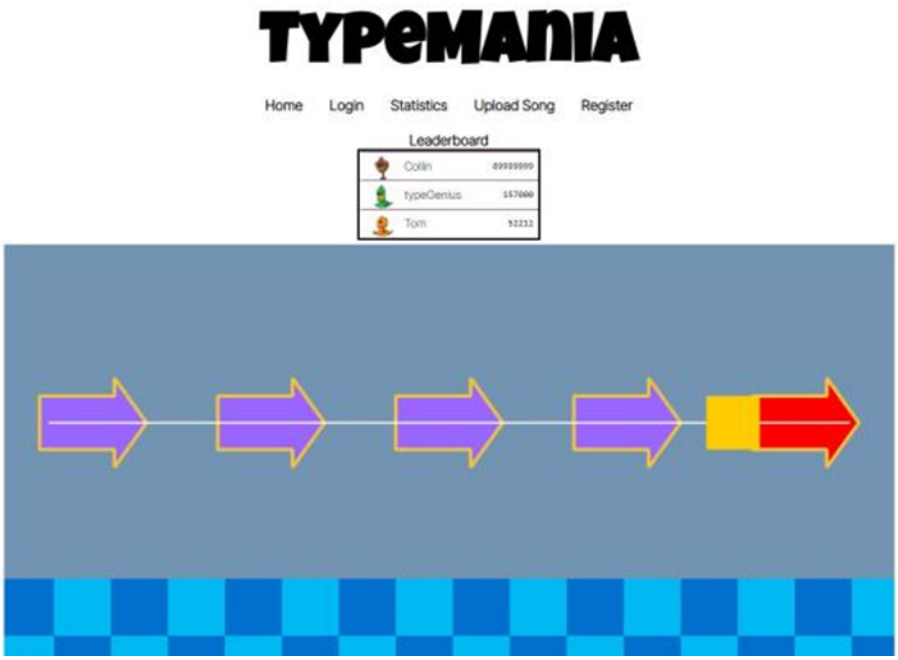

Figure 11: TypeMania Landing Page and In-Progress Gaming Component

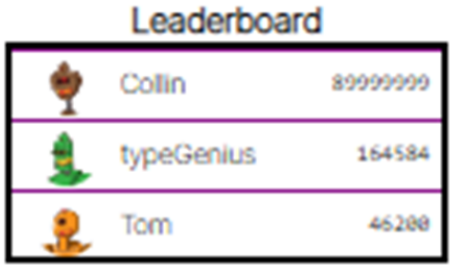

Figure 12: TypeMania In-Progress Leaderboard Component

The game component shown on the landing page is also an in-progress element. The presence of the game component will be consistent for both guest and registered users, but registered users will have a more customized game user interface, which will promote the key design principle of flexibility and efficiency of user by enabling customizable experiences for expert, or registered, users (Nielson, 2020).

The gameplay scene is not yet implemented in the gameplay component. Slight changes were made in the gameplay scene, see Figure 13, that will improve the user experience and ease-of-use once implemented into the game component. First the combo section from the first design is removed from the bottom left corner to create a more simplistic component that will not be as overwhelming for the user. Additionally, the flow of notes to the hit zone have been flipped to follow a left to right sequence, which will be more helpful to those of languages that read left to right. This flow of notes may be a customizable option in future installments. Notice the target key notes are shown in the blue squares, which will move to the hit zone, where the user must type at that time. The scene also features the score and accuracy similar to the prior design. All of these aspects of the gameplay component improve user experience providing real-time statistics during gameplay.

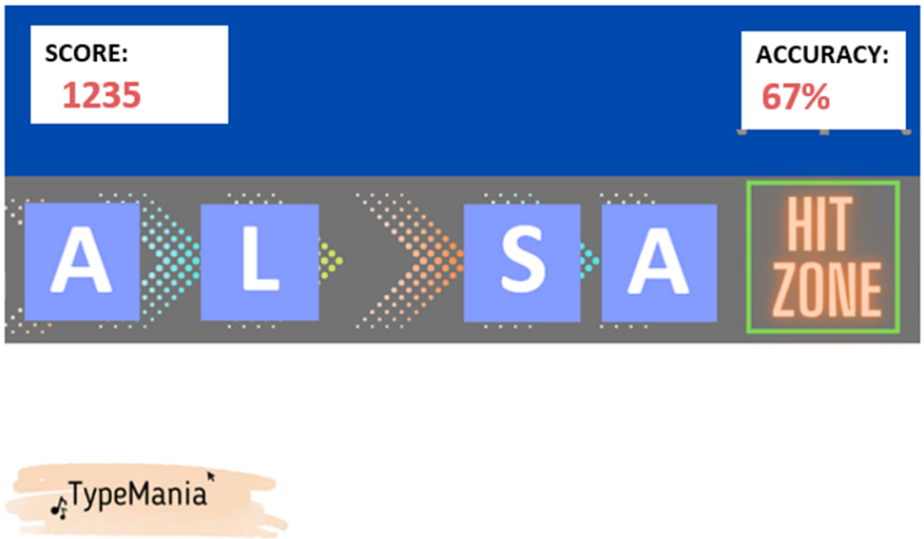

Figure 13: Gameplay Scene

The implementation of the gameplay interface, statistics page, and upload song page are still in development. Currently, each of these elements will still follow the proposed design; however, they will likely be changed as development continues and implementation is underway. During this process, component usage flow is better understood and tweaked for efficiency and ease-of-use. Additionally, implementation barriers can also cause adjustments in design to ease the stages of development.

6. Design of Tests

6.1 Unit Testing

Test Case	Description
Deadlink Checker	This code will check for links that either do not work or lead to nowhere.
Home page loads	Code prompts for the homepage to output the user's screen.
Login works for existing users	Code attempts the login process of a user successfully in the system and they are successfully logged in.
Additional features are available to registered users.	Code checks that additional features are available to the user after login is successful.
Login error message prompts for invalid information entered	Code entered invalid user information into the system while logging in to ensure the error message prompt appears.
Statistics screen displays up to date information	Code checks the last time the statistics screen was updated. The last update should be the last time a gameplay was successful.
Statistics screen appears after gameplay automatically	Code checks the statics screen is prompted to the user after gameplay is complete.
Statistics information changes depending if the the user is a guest or a registered user	Code checks the information available to registered users only appears after a user has successfully logged in.
Music functions work during gameplay	Code checks the music features played during gameplay.
Registration creates a new user	Code attempts registering as a new user with randomized information to ensure a new user can be created. However after the unit testing is complete the information will be deleted.
Registration prompts an error message if the	Code attempts registering as a new user

user is pre-existing	using random information.
Gameplay function displays working background	Code attempts to enter gameplay as a user ensuring basic gameplay functions are working.

Table 12: Test Cases and Descriptions

6.2 Testing Coverage

Test Case	**Testing Coverage**
Deadlink Checker	This will check every link in the program to ensure there are not any deadlinks. The deadlink checker will be automated, with written code to check all the links. As the amount of individual links is hard to approximate before the completed project, the number will undoubtedly be large between 50-100 making it very time consuming to perform manually.
Home page loads	This will check to make sure the home page loads properly for the user. The code will be directed to the home page link and ensure no error codes are returned.;l
Login works for existing users	Testing will check the database for pre-existing information and ensure the inputted information matches the information in the database. Only then will the user access additional features of TypeMania.
Additional features are available to registered users.	Testing will ensure the appropriate links are successfully loaded onto appropriate menus and work for the user after successfully logging in.
Login error message prompts for invalid information entered	This will check the error message after a user inputs the wrong login information. The system will be looking for typical error codes from the system, and ensure the pop up works correctly via hyperlink.

Statistics screen displays up to date information	Testing checks and outputs to the programmer the last time successful gameplay was completed and the last time the statistics were updated for the individual user and all users of TypeMania.
Statistics screen appears after gameplay automatically	The system will ensure, via hyperlink, the statistics screen automatically displays after successful gameplay. This primarily features the leaderboard for said song the user finished with.
Statistics information changes depending if the the user is a guest or a registered user	After a user has successfully logged in, the system will check that the appropriate links are activated so the user can access those features. The testing will also check those features are not available to a guest user because there would not be enough information to make said links useful to the player.
Music functions work during gameplay	Testing will play music so the user can hear and utilize the song they chose to play with. This includes volume at an appropriate level and the correct song being played.
Registration creates a new user	Testing ensures duplicate information is not accepted into the system and stores successful registration information in the appropriate database.
Registration prompts an error message if the user is pre-existing	When a user attempts to register with information that is already in use an error message is prompted. This will be tested via activating the correct hyperlink.
Gameplay function displays working background	Testing ensures the appropriate background is displayed for the user. This includes loading the correct screen with the correct colors, framework, and display options.

Table 13: Test Case and Testing Coverage

6.3 Integration Testing

Bottom up integration will be applied to TypeMania, we will test the smaller functions and proceed to testing the bigger picture progressively. First, all of the unit tests will be run, therefore if there are any bugs they will be easier to localize. The unit tests were designed to focus on the main functions of the application and if any tests need to be added as testing begins that can be done. After unit testing is successfully completed, the tests will become more broad. Being that TypeMania is a smaller application compared to major corporations there are few steps to working up the ladder. The second level of testing will include: registration, logging in or out, song upload, testing each individual statistic as a whole function, and testing each individual setting as a whole function. The last level of testing will include: statistics, settings, login, registration, and gameplay as those are the main headings in the menu. Running the entire program as a whole at one time and experimenting during gameplay will be the final test to ensure the user experience is of utmost quality.

Project Management

Merging the Contributions from Individual Team Members

While compiling the final copy of the report from individual team members, there weren't any major or significant issues faced. There were a few minor issues while exporting the report into PDF format. In PDF format, the page breaks were ignored and the page numbers didn't match the numbers in Table of Contents. This issue was tackled by inserting blank lines at the top of the page.

Project Coordination and Progress Report

Use case & functional feature implementations (as of Oct 16th, 2022).

Use Case	**Status**
UC-1 Guest User Page	In Progress
UC-2 Register	In Progress
UC-3 Registered User Page	Incomplete
UC-4 Typing Speed Selection	Incomplete
UC-5 View Score	Incomplete

UC-6 Music Selection	In Progress
UC-7 Play Game	In Progress
UC-8 Create a Songmap	Incomplete
UC-9 Uploading Music	Incomplete
UC-10 Style Choices Selection	Incomplete
UC-11 Leaderboard Display	In Progress
UC-12 View Statistics	Incomplete
UC-13 Login	In Progress

Table 14: Use case & functional feature implementation status

Project Management Activities

Activity	**Description**
Team Meetings	A voice chat between all team members is held each Saturday. Discussions center around tasks, assignments, deadlines, and other member responsibilities.
Shared Drive	The accumulation of the team's documents in one centralized location.
TypeMania Repository	The collaborative codebase of the project that all members contribute their task portions to.
Submission Collation & Proofreading	Prior to the deadline of a written assignment, the final draft is collated by one member and emailed out to the team for one last chance to spot and correct mistakes.
Researching & Teaching	The team will use many tools that they're not familiar with. If one member comes to understand a tool more quickly than the others they are encouraged to share their resources and understanding.

Table 15: Project management activities

Plan of Work

The plan of work after the submission of report #2 is not unlike that of report #1. We will continue pushing for the core elements of the game to create a workable prototype. After the gameplay is completed we will move into a scoring and feedback system. The focus will then be on content creation with the level creator. The remaining features and optimizations will come as time allows. See the charts below for more details.

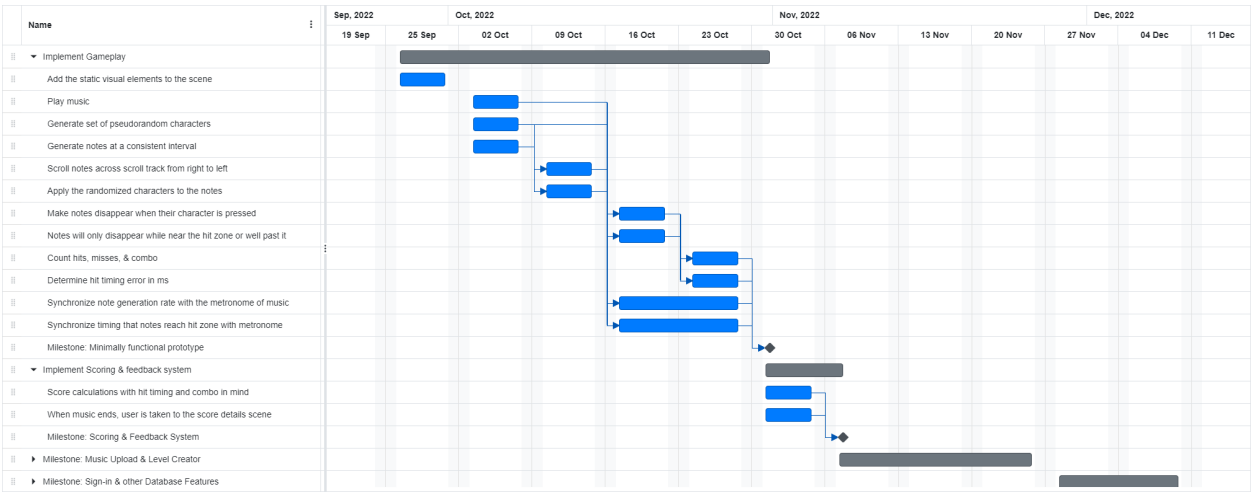

Figure 14: Plan of work - Project roadmap. [Full size image here]

Milestone	**Projected Accomplishment Date**
Minimally functional prototype (deployment & gameplay)	Oct 30th, 2022
Scoring & feedback system	Nov 5th, 2022
Music upload with level creator	Nov 25th, 2022
Sign-in & other database features	Dec 8th, 2022

Table 16: Projected milestones

Breakdown of Responsibilities

Responsibility	**Member**
Controller module	Alex
Database module	Nargis
Interface module	Heather
Game module	Shaka
Validator module	Nargis
Integration coordination	Alex
Integration testing	Shaka

Table 17: Breakdown of responsibilities

The responsibility matrix above should be viewed as a product ownership listing. That is, the responsible member is not the only one implementing classes for their module; Instead, they are the primary coordinator for and most knowledgeable member regarding their module.

Sources

Jha, R. (2021, January 4). *GRASP Design Principles*. mySoftKey. Retrieved October 9, 2022, from https://www.mysoftkey.com/architecture/grasp-design-principles/

Above resource was used in coordination with the textbook.

Nielsen, J. (2020, November 15). 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. Retrieved October 16, 2022, from https://www.nngroup.com/articles/ten-usability-heuristics/