**Mobile Apps Final Specifications:**

**Soundboard Quiz**

Luke Aschenbrenner

Department of Mathematics and Computer Science

CSCI-3980 Mobile Apps

Professor Nicolaas tenBroek

May 7, 2023

**Application Functionality**

There are many functions that this app can perform to fulfill its goal as a Soundboard Quiz. On the user-facing side, the three buttons that appear when the user first opens the app are “Resume”, “New Game”, and “High Scores”. Each of these three buttons correspond with a separate action, all of which segue into a new view when applicable. The “Resume” button’s functionality is straightforward: if a user was in the middle of a game and pressed the home button to exit the app, if the app is about to be removed from RAM, it will save the state of the game. When the user re-opens the app, the resume button will act as a method to continue where they left off. If no game was saved, the resume button will display an alert notifying the user that there is no game in progress. When “High Scores” is pressed, the app displays a basic UIView with labels designating the top 5 scores from all game modes, specifying the score and each game mode for each entry. By pressing “New Game”, the app queries the database for available Sound Categories to choose from and displays them in alphabetical order. This is displayed using a table view. Each sound category object contains a name, a list of sounds that correspond with the category, and a list of images that correspond with the sounds.

When the user selects a category, the navigation controller communicates with the destination view controller about the category selected and continues to segue to the main game view. Most of the core functionality of Soundboard Quiz is located inside this main game view. The main game view contains 5 major components: The navigation bar, a score label, a shuffle button, and two container views. The navigation bar allows the user to exit back to the category selector and also displays the name of the current category in the header. The score label displays the current game score for the session, initializing to zero. The shuffle button randomizes the content displayed in the two container views, using data from all the sounds present in the current sound category. Finally, the two container views contain collection views, with the left responsible for displaying sound boxes, and the right for displaying image boxes. Both collection views display four items (“boxes”) at a time in a vertical stack format. The left, sound box collection view, has user interaction enabled. The user can interact with the boxes in 2 ways: firstly, the user can choose to tap on the box. When this is done, the sound associated with the box will begin to play and the play icon will change to a stop icon. The “plays left” counter on the box decreases on each play. When the counter reaches zero, the box becomes grayed out and the user is no longer able to tap it. The second way in which the user can interact with the box is by dragging it. When the box is dragged onto one of the four image boxes, one of three things happens. If the sound box was dropped on the correct corresponding image, both the image and the box become gray and can no longer be interacted with. In addition, the score increases by one. If the box is dropped on an area with no image box below it, then it returns to its original position. But, if the box is dropped on a wrong image box, then the app segues to the Game Over view, which displays the user’s final score. If a user successfully matches all four sounds with their corresponding image boxes, then an animation plays. In this animation, the eight boxes slide up and make room for eight new boxes, chosen randomly from the list of sounds in the sound category. The game continues until the user makes a mistake and loses the game.

In addition to the core game functionality, some effort has been made to support users with disabilities. Specifically, blind users should be able to play this game without needing to view the image boxes. When accessibility features are turned on, the app will read out many details, such as how many plays and shuffles are left, the name of the category, and most importantly, the title of each image. Although this could be viewed as cheating, it is better for blind users to have a slight advantage in the game by always being able to identify the items in the images than not allowing them to play at all.

**Data Storage and Retrieval Methodologies**

Much of the app’s functionality is accomplished in the backend. When the app is first launched, a local SQLite database is generated from a Property List specification. This happens in the AppDelegate class, where an Persistent Container is created to be used in the Core Data stack present in this app. This container is then called upon to perform the data preloading in a background thread. A file called “PreloadedData.plist” is loaded into memory as an array of arrays, of which contain Dictionaries and Strings. The Strings represent the name of the category, and the Dictionaries hold the name of each sound and its associated image/sound file name. This data is converted into a core data compatible format by instantiating custom Core Data classes named SoundCategory and Sound respectively. Since the database is relational, a relation is created as a one-to-many between the SoundCategory object and its corresponding Sounds. These objects are referenced later at various points in the app. The app also extends the database object functionality to allow for saving and retrieving the state of the game upon termination. By choosing “Category/Extension” under the data model file for the Sound object, I created an extension for the Sound class that conforms to the Codable (Encodable/Decodable) interface. This allows the app to convert sound objects to JSON and vice versa, which is necessary for storing the game’s current state on disk using UserDefaults. To test this behavior, the app has a custom build property that forces the app to terminate and clear itself from memory on every home button press. This can be [easily removed](http://pinkstone.co.uk/how-to-kill-your-app-when-it-enters-background-mode/) to revert back to normal behavior.

UserDefaults are also used to store the top five high scores across game sessions. Each time a game over occurs, the app compares the newest score to the top 5 high scores, and, if it is greater than or equal to any of them, it places them in the list. Finally, the app contains a lot of raw data files which the database references as Strings. According to some internet resources, it’s not a very good idea to store binary files in a CoreData database. Instead, sound files are stored in wav or mp3 format inside of their own asset folder. The images are stored inside of an .xcassets folder. One of the downsides of this approach is that I am not able to use subfolders to organize the data. With a very big collection of categories, it could become cumbersome to maintain such a large list of unsorted files. One temporary fix would be to prefix each file with its category, which may prove useful in the short term.

**Application Screenshots**

| This is a view of the app’s icon from the home screen. | This is a custom LaunchScreen storyboard shown on initial app launch. |
| --- | --- |
| This is the main menu view of the app. | This is the high scores view, with a few sample sessions included. I really couldn’t get that ninth national anthem! |
| This dialogue box appears when the app wasn’t closed in the middle of a game, so there is no session to resume. | This is the view that appears when the New Game button is pressed. Currently, only the views with a \* beside them have actual sound files associated with the category. |
| This is the view that appears when a category is selected. | The play button was pressed on the top left box. |
| This is an example of dragging the top left box over an image. The green plus icon means it is a valid image, but it may not be the right one. You can also see the third box is gray, signifying that we ran out of plays for that box. | The drums were a correct match for that sound, so now user interaction is disabled for both boxes. This is signified by a lighter gray color. |
| A user is dragging another sound box over the corresponding image. This was repeated until all the boxes were matched (score equals four). | When all the boxes were matched, the new box animation played and the boxes scrolled up off screen. |
| A screenshot of the new randomized boxes. Only the trombone is a duplicate from last time. | Another animation example. |
| The “Game Over” view, which displays when a user drags a box and drops it over the wrong image. | The new high scores for our current game. I got first place! |