Famus

Design Document

Team 7

Ye Chen  
Di Liu  
Peng Liu  
Tao Tian  
Yusen Zhang  
Joshua Childress
1. Purpose

1.1 Non-Functional Requirement

1.2 Functional Requirement

2. Design Outline

2.1 High-level overview of the system

2.2 Detailed Overview of the Client/Server/Database Architecture

2.3 Broad Overview of Sequence of Events

2.4 Overview of the Different States of the System

3. Design Issues

3.1 Functional Issues

3.2 Non-Functional Issues

4. Design Details

4.1 Class Diagram

4.2 Descriptions of Classes and Models

4.3 Sequence of Events When User First Starts Application

4.4 Sequence of Events When User Creates a Recording

4.5 Sequence of Events When User Comments on a Recording

4.6 UI Mockup
1. Purpose

A lot of people think they have good singing skills but cannot find anyone to appreciate their voices. Many people have confessions to make but are afraid of judgement from their friends. People sometimes seek advice, but are too embarrassed about their situations to ask. Even though there are karaoke apps and anonymous confession apps, there is no app that allows users to share their voices and thoughts with people.

We believe that voice is a more intuitive and intimate way to share and connect with people. By allowing users to share via voice and giving them the option to add special sound effects such as beats or background music, our product not only shortens the distance between people but also provides a way to inspire the person sharing as well as those they are sharing with.

The purpose of our application is to enable users to share their audio recordings, or even just to speak their confessions and thoughts out loud, and see what people think of them.
1.1 Non-functional requirements

As a developer,

- I would like to easily build project sources files into a deployable bundle.
- I would like to easily deploy executable bundle on a iDevice and run it.
- I would like to run UI tests and receive feedback as whether the test passed or not.
- I would like to get the latest version of the app from a source control.
- I would like the app to be able to run on most iDevices with a variety of system versions.
- I would like the app to minimize data communication between the app and the server.
- I would like the app to require minimal effort for users to understand and use it.
- I would like the app to be as smooth as possible in terms of both UI and data communication.
- I would like the app to use encrypted protocol to make communications between clients and the server.
- I would like the server to be secure in the way that non-authorized requests are denied.
- I would like the server to be able to handle at least a million users.
1.2 Functional requirements

1.2.a Users can browse and vote on recordings using the application.

As a user,

- I would like to see all of the existing recordings around.
- I would like to vote on each recording and see the vote totals instantly.
- I would like to change the previous voting at any time.
- I would like to comment on each recording with text or emoji, and see all the comments.
- I would like to remove previous comments.
- I would like to rank recordings by time or popularity.
- I would like to have notifications if my posts have any updated activity.

1.2.b Users can record and process recordings.

As a user,

- I would like to pause the recording and resume it later.
- I would like to see the sound waves of my voice while recording, so I created high quality recordings.
- I would like to add sound effects instantly during recording.
- I would like to edit my recording before posting to the public.
• I would like to add titles/hashtags to my recordings.

• I would like to save recordings I created to local drive so I can share with others later.

1.2.c Users can look up existing recordings using a database.

As a user,

• I would like to search recordings based on titles, user name of the user who uploads it, and hashtags that were created upon uploading.

• I would like the search bar to auto-suggest possible matching entries

• I would like to see the history high ranks for the past week.

• I would like to see all of the recordings I have created and posted.

• I would like to delete my posted recordings.

1.2.d Users can customize the application.

As a user,

• I would like to change background pictures to any of the pictures from my photo library.

• I would like to change global tint color to any color via RGB value.

• I would like the graphical interface to be intuitive.
1.2.e Users can navigate the application with an innovative design.

As a user,

- I would like to see this app have a cool icon.
- I would like to see this app have a cool loading bar/icon.
- (if time allows) I would like to see the app has a fancy splash screen or even meaningful videos.
- (if time allows) I would like to see fancy UIs that runs smoothly.

1.2.f The back-end will be hosted on AWS.

As a developer,

- I would like the database to store all the recordings based on unique user identifier.
- I would like the server to communicate between clients and the database.
2. Design Outline

This project will be an application that allows users to publicly post customized audio recordings and see the audio recordings of others within the same geographic location, as well as to vote and comment on those recordings. This application will use the Client-Server model, with one server handling connections from multiple clients. The server will implement the Model-View-Controller (MVC) pattern. The server will access a database where all recordings, recording data and user profiles are stored.

1. **Client**
   a. The client will be where the user's interaction with our system will be.
   b. The client will display an ordered list of recordings and their details.
   c. The client will show UI features where the user can search, create or vote on recordings.

2. **Server**
   a. The server will handle all traffic between the application and the database.
   b. The server will transfer the names, descriptions and identifiers of the recordings from the database to the application at startup.
   c. The server will retrieve audio files from the database when the application requests them for playback.

3. **Database**
a. The database will store all of the recordings and their details for this application.

b. Typical data will be audio files, comments and vote totals.

2.1 High Level Overview of the System

This project will have many clients connected to the server at the same time. The clients will send requests to the server when they need user data, recording data or recordings. The server will request the appropriate data from the database, sort the list and return it to the clients. When a specific recording is requested for playback, the server will request the audio file and data and send it to the client. Requests to sort the recording list without updating recordings or data will be handled by the client.
2.2 Detailed Overview of the Client/Server/Database Architecture

The client and server communicates using HTTPS requests. Client sends POST method when it is posting new recordings, new comments, or making new upvote or downvote. Server that is hosted on AWS then queries the specific data that can satisfy the request from the database using APIs provided by our database provider Parse. With the data that server acquires from the database, a response object in the form of JSON is generated and will be sent back to the client. When the client gets the response, it updates the model accordingly.
2.3 Broad Overview of Sequence of Events

This diagram shows a typical sequence of events when using the application. Requests that can be completed by the client will be handled by the client. Requests to the server to read or write recordings or data will be sent to the server and added to the database. Any changes to the database will return a list of recordings and data which will be sorted by the server and returned to the client. Requests by the client for playback of a recording will be sent to the server and the specified file and its data will be requested from the database and returned to the client.
2.4 Overview of the Different States of the System

This diagram describes the various states the system can be in. When the user opens the application, the application gets into the Main UI state. The user can then go to a specific state when pushing the corresponding button. The application will go back to the Main UI State when the user pushes the back button. Note that when user clicks on the home button of the iDevice he/she is using, the application will go into inactive state. When the user decides to resume the application, the application will go back to the last state it is in before going to inactive state. However, in order to make this diagram clean, the inactive state shows up as if it is only interconnected to the Main UI state.
3. Design Issues

The followings are some of the design issues that are made during the course of the planning process.

3.1 Functional Issues:

Functional Issue #1 - What kind of in-recording sound effect should we provide

- Option 1: simple fix noise and boost volume effect
- Option 2: simple fix noise and boost volume effect with special sound effect such as beats, suspense sound, or heartbeat sound.
- Choice: we choose option 2.
- Discussion: Special sound effect such as beats, suspense sound often sparkle users’ creativity. By providing a tool for users, we hope that the community is filled up with fun and innovations.

Functional Issue #2 - Whether or not to let user comment directly on the historic high rank’s page

- Option 1: Yes
- Option 2: No
- Choice: Yes
- Discussion: If no, then user will have to click on the comment button of a specific cell in order to get to the comment view, and then make comments. If yes, by simply adding a expandable keyboard in the page, user can directly comment on
the view, and if user wants to comment on a specific comment that is made by others, click on the comment button in the cell that contains the comment, a auto-generated quote will be pasted in the text field.

Functional Issue #3 - Whether or not to provide historic high ranks page

- Option 1: Yes
- Option 2: No
- Choice: Yes
- Discussion: By providing a historic high ranks page, good creations are not lost fast. Good creations always sparkle discussions, and discussions do not end in a day. We want to provide as many interactions among users as possible, so the app stays refreshed and interesting. For those who gets on the historic high ranks, it is also encouragement for them to keep creating interesting recordings. In that way the community for this app stays entertained. By providing the historic high ranks page, we provide a ‘market’ for those who wants to get flash lights, and a ‘need’ for those who wants to get entertained.

3.2 Non-Functional Issues:

Design Issue #1 - What IOS development language will we use for the app?

- Option 1: Swift
- Option 2: Objective - C
- Choice: Objective - C is our choice for the application.
• Discussion: Three members of ours have zero IOS development experience. One has beginner Objective-C experience. One has beginner Swift experience and one has advanced Objective-C experience. People with Objective-C experience are majority. Another reason is that Swift is a relatively new language so it doesn’t have as many APIs or libraries that support Swift as Objective-C. Therefore for the sake of application quality we decided to choose Objective-C as our programming language.

Design Issue #2 - What backend web service are we going to use?

• Option 1: Develop the central database by ourselves.
• Option 2: Use backend providers.
• Choice: Using a backend provider is our choice for the application.
• Discussion: Option 1 has a lot of heavy time investment. The time developing a back end just about doubles the front end work involved. Also it is heavy skill investment. Since a majority (five out of six) of the team members are inexperienced IOS developers, we don’t think it is a good strategy to invest huge amount of time on backend development. The third is scalability issues. The nature of iOS apps and the App Store means you’ll never know whether your app will only have limited usage, or become a huge hit with millions of users. Developing the back-end so it scales efficiently with usage is quite a challenge. Option 2 allows us to have a lot of services. First of all, it gives us an easy way to take data from
object format, and store/retrieve it from the central database. Second, most backend services will provide file storage that we can use to associate a file with a custom object. Third, we will need to implement the geolocation service for the app. Backend services often provide the ability to tag the objects with a specific location so the user can make queries on objects based on location. Forth, almost of all the Backend-As-A-Service offerings allow developers to create their own users, and most of the users login via Facebook or Twitter so that the user doesn’t need to create a new account. This feature is very important since we are going to use the Facebook login. Thus we decided to use a backend provider instead of developing the central database by ourselves.

Design Issue #3 - What Backend-As-A-Service are we going to use?

- Option 1: StackMob
- Option 2: Parse
- Option 3: Appcelerator Cloud Service
- Choice: Parse is our choice for the application.
- Discussion: For StackMob, one disadvantage is that if we want to serve up binary assets, like the images in our app, we will need an Amazon S3 account. You can’t maintain binary assets directly in StackMob. Their free plan is one of the best in the market, it’s completely free for basic features and you can purchase new features in their MarketPlace. Parse has one
of the best free plans out there. The documentation is the easiest to understand, and the service is one of the easiest to work with, especially for beginners like us. Parse has sample projects that we can download, as well as custom UIViews that we can add to our project, such as Facebook and Twitter login views. Additionally, there is a large collection of 3rd party libraries offered for use with the service. For Appcelerator, the development complexities (and costs) rise more than proportionally to application complexity increases. The more complex our application becomes, the more often we’ll have to deal with technical issues (random crashes, weird behaviors, annoying bugs, etc.). Parse seems to have more advantages than the others. Thus we decided to choose Parse as our backend database.

Design Issue #4 - How should Users login to the app?

- Option 1: users don’t need to sign up for using the app.
- Option 2: users can choose to sign in with their Facebook account.
- Option 3: users can either continue using Facebook account or anonymously.
- Choice: Option 3 is our choice for the application.
- Discussion: At first we wanted the users to use the app anonymously because we thought that anonymous recordings would be fun and interesting. However, we then realized that anonymous recordings would
make it hard for us to keep track of the user's own recordings. There is a way to solve this issue but due to the time limit of this project we don't think we will be able to solve this issue. Then we come up with a idea that users can sign in with their Facebook accounts and also sign in anonymously. For those users who sign in with Facebook accounts, they can keep track of their own recordings and see how many upvotes, downvotes and comments they have got but people can see their profile picture on their posts. If they choose to use the app without signing in, they can post recordings anonymously but they can not see the history or record of their anonymous posts. Having two login options will suffice those users who would like to be seen and heard by others and users who are not willing to reveal their online identities.

Design Issue #5 - Should we use HTTP or HTTPS for our app?

- Option 1: HTTP
- Option 2: HTTPS
- Choice: Option 2 is our choice for the application.
- Discussion: HTTP is HyperText Transport Protocol which is the way data gets passed back and forth between web servers and clients. HTTPS is the secure version of HTTP and the 'S' at the end of HTTPS stands for 'Secure'. With HTTPS, communications between your browser and the website are encrypted. If implemented in our app, private information
such as log-in information and recordings will be much less likely to be stolen; for this reason, we chose to utilize HTTPS for our app.
4. Design Details

4.1 Class Design:
4.2 Descriptions of Classes and Models:

A list of controllers we use: FirstViewController, SecondViewController, ThirdViewController, PostNewRecordingViewController, and CommentViewController. Each of these controllers are a subclass of UIViewController, a class implemented in Apple’s provided framework UIKit.

- **FirstViewController**: is composed of a table view, a search bar, a navigation bar, and a tab bar. It displays the list of recordings associate with user’s geolocation in the form of a table, enabling the user to filter the list from the table based on tags, recording names, or creator’s username.
  - **UITableView**: is a class of UITableView, is composed of cells. It is the view that displays each recording from the list in every cell.
  - **RecordingTableCell**: is a sub-class of UITableViewCell, is composed of few UILabels and a few buttons. This is where the user can interact with a specific recording.
  - **UIDebuggers**: By tapping on the play button in the cell, the user can play the recording that is contained in the cell. By tapping on the upvote button, a POST HTTP request is sent to the server asking to upvote the recording in the cell by one. By tapping on the downvote button, a POST HTTP request is sent to the server asking to downvote the recording in the cell by one. By tapping on the comment
button, a `CommentViewController` instance is modally displayed, where user can make comments.

- **UILabels**: A label with the total number of comments for the recording in the cell is displayed beside the comment button. A label with the result of total upvotes minus total downvotes is displayed on the very right side of the cell to indicate general response from other users. A label with the title of the recording is displayed in a big font. A label with the description of the recording that was provided by the creator is displayed below the title label.

- **UISearchBarController**: is a class of `UISearchBarController`, Which uses right hand and left hand expression to evaluate the specific key in the `NSObject`, by implementing the delegation method, `UISearchBarController` will automatically reload table with the filtered result.

- **UINavigationBar**: is a class of `UINavigationBar`, which provides a button to dismiss the modal view when any views are modally displayed on the root view controller.

- **UITabBar**: is a class of `UITabBar`, which provides a way to switch between the three views paralleled exists and are controlled by instances of `FirstViewController`, `SecondViewController`, and `ThirdViewController`.

  - **UIBarItem**: is a sub-class of `UIButton`. Upon instantiation, the tabBar instance is inserted with three `UIBarItem`'s. Each `UIBarItem`
is assigned with an action that replace the current view and display another view that is controlled by one of the three view controllers accordingly. Upon instantiation, the default displayed view is the view that is controlled by FirstViewController.

- **SecondViewController**: is a sub-class of UIViewController. SecondViewController is composed of a UIImageView, a UITableView, and a few UILabels, and a UITextField. The purpose for SecondViewController is to display a histogram that shows historic highest vote recordings for the past week, and let user directly comment on each recording.
  
  - **UIImageView**: is a class of UIImageView. UIImageView will display the histogram using the drawing Apple provided method draw shapes using Bézier Paths. Then a few UIButton is displayed with each recording’s title on top of the histogram to identify each recording.
  
  - **UITableView**: is a class of UITableView, is composed of CommentTableViewCell. It is the view that displays each comments for the specific recording. The default recording is set to the first recording on the histogram.
  
  - **CommentTableViewCell**: is a sub-class of UITableViewCell, it displays the comment made by another user, as well as that user’s username. The comment UIButton on the cell will auto quote the comment content in the cell to indicate that the current user is making a comment for that comment.
 UITextField: is a class of UITextField, the purpose for this text field is to enable user to directly post on the page, when the text field is assigned to the first responder, text field will move up and make some space for the keyboard to expand. When the text field resign from the first responder, the keyboard will move down to disappear, and the text field will move down and back to its original position: bottom of the view.

- ThirdViewController: is a sub-class of UIViewController. ThirldViewController is composed of a UIImageView to show user’s profile, a UIButton to connect to FaceBook, and a UITableView to show all the recordings that this user has made in the past. The purpose for this view is to show everything that is related to the current user.
  
  UIButton: is a class of UIButton, this button is assigned with an action to request permission to connect current user profile with the user’s Facebook account. When this button is tapped, if the current device has Facebook installed, Facebook app will then open and asking for the permission. If Facebook is not installed, a web page will then be opened and asking the permission. After the permission is either granted or rejected, the app will then re-enter to the foreground.

- UITableViewCell: is a class of UITableViewCell, is composed of instances of RecordingTableViewCell, the purpose of this UITableViewCell is to provide an overview of the recordings that this user has made in the past.
**PostNewRecordingViewController**: is a sub-class of **UIViewController**, it is composed of a **UIImageView** that indicates user’s volume in waves, two **UITextFields** that let user input the title and description for the recording, a **UIButton** that starts the recording, and few **QPushButton**s for each special sound effects, a **UIButton** that post the recording. When user is recording, special sound effect buttons is enabled, each tap on the sound effect play the sound effect as well as record it in the current sound stream. The purpose of this view is not only to provide a way to record users’ recordings but also as well as some tools to improve the quality of their recordings.

**CommentViewController**: is a sub-class of **UIViewController**, it is composed of a **UITableView** that displays all comments associated with a specific recording, a **UILabel** for this specific recording’s title, a **UILabel** for this specific recording’s description, a **UILabel** for this specific recording’s creator, a **UILabel** for this specific recording’s created date, a **UIButton** to play this specific recording, a text field to comment on this specific recording or any comments that is commented on this specific recording.

- **UITableView**: is a class of **UITableView**, it is composed of instances of **CommentTableViewCell**, each **CommentTableViewCell** represents a comment that is made by another user on this specific recording.

- **UITextField**: is a class of **UITextField**, the purpose for this text field is to enable user to directly post on the page, when the text field is assigned to the first responder, text field will move up and make some space for the
keyboard to expand. When the text field resign from the first responder, the keyboard will move down to disappear, and the text field will move down and back to its original position: bottom of the view.

4.3 Sequence of Events When User First Starts Application:
4.4 Sequence of Events When User Creates a Recording
4.5 Sequence of Events When User Comments on a Recording
4.6 Mockup UI

The up and down arrow will change the voting (shown as the number in the middle). A second click will cancel the previous voting.

First View

The play button will become a pause button after clicked, so that the recording can be stop at any time. A progress bar will show around the button.

This mic will lead to the record view
Comment View
Recording View
Historical Rank View

Profile View

Copyright © 2015 CS307 Team 7. All rights reserved.