Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

Asaprappy Deployment Document

AsapRappy will be deployed on the Tomcat 9 server with MySql database version 5.1.46.

1.0 Push Application to Server

Import the project to the Tomcat 9 server, including the build, src, and WebContent directories.

2.0 Configure external files

Connect the MySql connector JAR file to the Tomcat server. This Java file is required to use JDBC to connect to a MySQL database Download Mysql-connector-java-5.1.46 from the MySql JAR archive. Move the JAR file to Asaprappy's lib folder. In the project's properties, add the JAR to the Java Build Path. Furthermore, add the JAR file as a deployment assembly.

Configure the MySql database. Import the database.sql script into the MySql database. Run the SQL script to create the database on the server. Run the script before running the application.

3.0 Perform Regression Test

Replicate the testing required in the testing document specifications. Run all testing code in the test directory, ensuring the project has successfully migrated to the server.

4.0 Launch the Application

Run the landing.jsp file in WebContent.

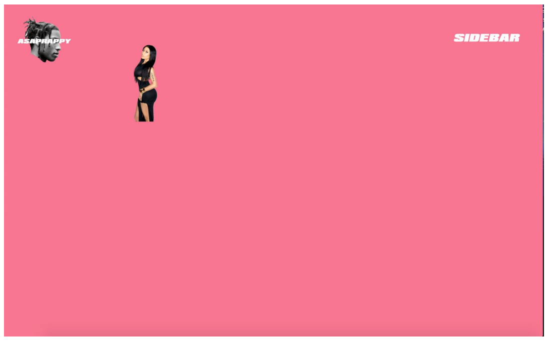
Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

AsapRappy Testing Document

Changes from 11/18/2018

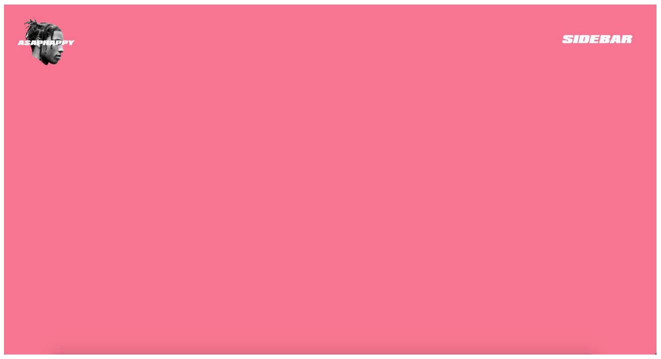
Test #	01 (White Box)
Test Description	Initial coordinates for the generated rapper images should be within the screen view.
Steps to run test	1. Run script, generate.js, to generate and validate screen coordinates. a. Generate.js will run the javascript function ShowImage() that creates coordinates for the images within a for loop. b. ShowImage() will generate coordinates with a given view screen width and height. c. These coordinates will be passed to ValidateCoordinates(x, y). d. ValidateCoordinates(x, y) will output "COORDINATES INCORRECT" for any coordinates outside of the screen dimensions.
Expected Result	The expected result is no output.
Actual Result	No output. All coordinates were within the given screen coordinate.

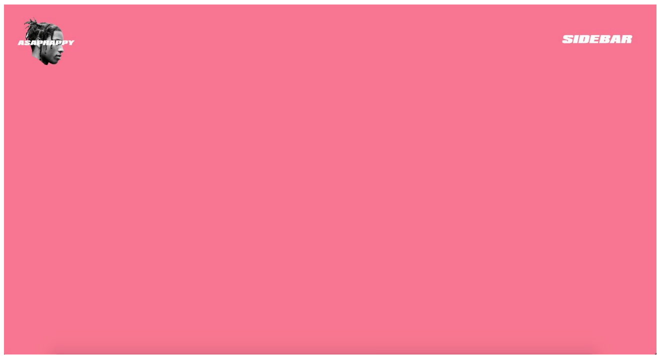
Test #	02 (White Box)
Test Description	Keypress (a-z, case insensitive) generates the correct image.
Steps to run test	1. Press any key without caps lock (a-z). 2. Verify rapper image corresponds to the pressed key. a. Visually check the rapper corresponds to the rapper image specified in the key-rapper map. 3. Press any key with caps lock on (A-Z). 4. Verify rapper image corresponds to the pressed key. a. Visually check the rapper corresponds to the rapper image specified in the key-rapper image map.
Expected Result	The correct rapper image should appear on the screen.
Actual Result	The correct rapper image appears on the screen.

Screenshot	 Nicki Minaj corresponds to key press s and S.

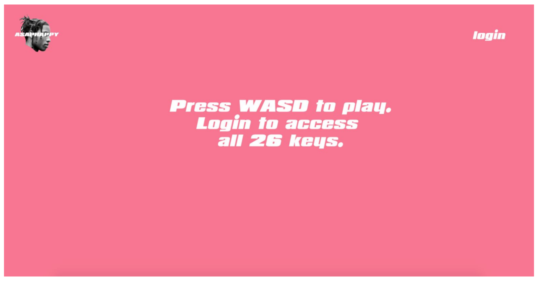
Test #	03 (White Box)
Test Description	Keypress (a-z, case insensitive) generates the correct sound.
Steps to run test	1. Press any key without caps lock (a-z). 2. Verify rapper sound corresponds to the pressed key. a. Visually check the rapper sound corresponds to the labeled sounds specified in the key-rapper sound map. 3. Press any key with caps lock on (A-Z). 4. Verify rapper sound corresponds to the pressed key. a. Visually check the rapper sound corresponds to the labeled sounds specified in the key-rapper sound map.
Expected Result	The correct rapper sound should play.
Actual Result	The correct rapper sound is played. For example, key presses s and S correspond to Nicki Minaj's "Purr" sound, labeled "nickipurr.mp3".

Test #	04 (Unit Test)
Test Description	Pressing the login button prompts the Google Sign In API. If the user is already signed in, the Google API automatically redirects to index.jsp.
Steps to run test	1. Press the login button. 2. Log in through a Google email. 3. Go back to the landing page without logging out. 4. Press the login button again.
Expected Result	A Google Sign In pop up should appear and the page should automatically redirect to the logged in page.

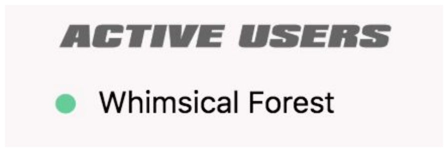
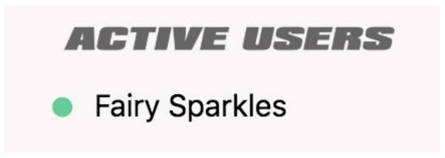
Actual Result	The Google Sign In pops up and immediately redirects to the logged in page.
Screenshot	 Page after redirecting.

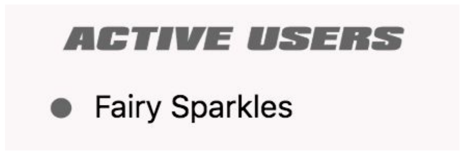
Test #	05 (Unit Test)
Test Description	Pressing the login button prompts the Google Sign In API. If the user is not already signed in, the Google API automatically redirects to index.jsp.
Steps to run test	1. Press the login button. 2. Log in through a Google email.
Expected Result	A Google Sign In pop up should appear, and prompt the user to sign in to a Google email. After signing in, the page should redirect to the logged in page.
Actual Result	The Google Sign In pops up, and after logging in with an email, the page redirects to the logged in page.
Screenshot	 Page after redirecting.

Test #	06 (Unit Test)
Test Description	Pressing keys that are not a-z (case insensitive) will not generate an sound/image combination.
Steps to run test	1. Press all keys that are not a-z or A-Z.
Expected Result	Nothing should happen.
Actual Result	No sound/image combination is triggered for keys that are not a-z or A-Z.

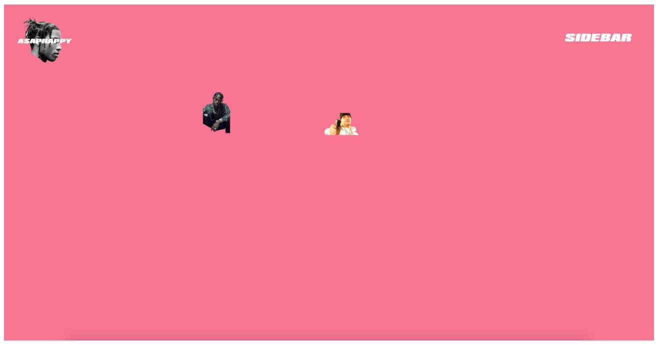
Test #	07 (Unit Test)
Test Description	The logout button signs the user out of the application by redirecting them to the landing page.
Steps to run test	1. Press the logout button
Expected Result	The user is automatically signed out after pressing the logout button, and the page is redirected to the signed out page.
Actual Result	The user is signed out and the page is redirected to the signed out page.
Screenshot	 Press WASD to play. Login to access all 26 keys. The redirected to signed out page after signing out.

Test #	08 (Unit Test)
Test Description	Active users have a green dot displayed next to their name.
Steps to run test	1. User 1 logs in. 2. User 2 logs in. 3. User 1 checks the sidebar. 4. User 2 checks the sidebar.
Expected Result	In User 2’s sidebar, User 1 should have a green dot next to their

	username. In User 1's sidebar, User 2 should have a green dot next to their username.
Actual Result	Both active users have a green dot next to their username.
Screenshot	 

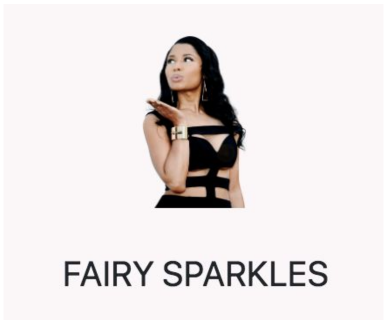
Test #	09 (Unit Test)
Test Description	Inactive users should have a grey dot displayed next to their name.
Steps to run test	1. User 1 logs in. 2. User 2 logs in. 3. User 2 logs out. 4. User 1 checks sidebar.
Expected Result	In User 1's sidebar, User 2 should have a grey dot next to their username.
Actual Result	User 2 has a grey dot next to their username.
Screenshot	

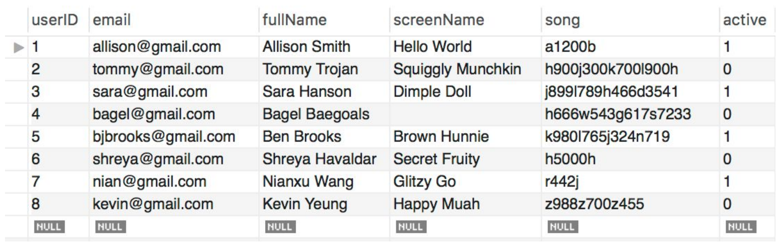
Test #	10 (White Box)
Test Description	Pressing multiple keys at once should generate the correct corresponding images and sounds for each keypress.
Steps to run test	1. Press multiple keys simultaneously without caps lock (a-z). 2. Verify rapper images and sounds correspond to the pressed keys. a. Visually check the rapper images correspond to the rapper images specified in the key-rapper image map, and the sounds correspond to the rapper sounds specified in the key-rapper sounds map. 3. Press multiple keys simultaneously with caps lock on (A-Z). 4. Verify rapper images and sounds correspond to the pressed

	keys. a. Visually check the rapper images correspond to the rapper images specified in the key-rapper image map, and the sounds correspond to the rapper sounds specified in the key-rapper sounds map.
Expected Result	Multiple images and sounds should appear in sync with the keypresses. Each image and sound should correspond to the correct rapper image in the key-rapper image map.
Actual Result	Multiple images and sounds appear in sync with the keypresses. Each image and sound corresponds to the correct rapper image in the key-rapper image map.
Screenshot	

Test #	11 (Black Box)
Test Description	Users should automatically be given usernames after logging in.
Steps to run test	1. Log in. 2. Check the sidebar for a username.
Expected Result	A user should be given a pre-generated and unique username.
Actual Result	After logging in, the sidebar displays a unique whimsical username. After logging in again with a new email, the sidebar displays a different username.

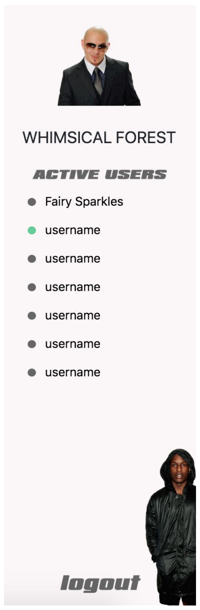
Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

Screenshot	

Test #	12 (White Box)
Test Description	Every user should have some type of song string in their database.
Steps to run test	1. Write a function to output all the song strings for every user in the database.
Expected Result	All the song strings will be a song with keypresses and waiting times or an empty string “”.
Actual Result	Same as expected result. All the song strings consisted of character combinations with waiting times in between of the default empty string “”.
Screenshot	

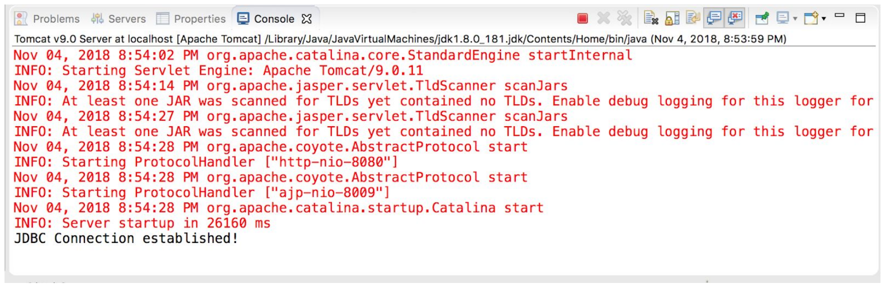
userID	email	fullName	screenName	song	active
1	allison@gmail.com	Allison Smith	Hello World	a1200b	1
2	tommy@gmail.com	Tommy Trojan	Squiggly Munchkin	h900j300k700l900h	0
3	sara@gmail.com	Sara Hanson	Dimple Doll	j899l789h466d3541	1
4	bagel@gmail.com	Bagel Baegoals		h666w543g617s7233	0
5	bjbrooks@gmail.com	Ben Brooks	Brown Hunnie	k980l765j324n719	1
6	shreya@gmail.com	Shreya Havaldar	Secret Fruity	h5000h	0
7	nian@gmail.com	Nianxu Wang	Glitzy Go	r442j	1
8	kevin@gmail.com	Kevin Yeung	Happy Muah	z988z700z455	0
NULL	NULL	NULL	NULL	NULL	NULL

Test #	13 (Stress Testing)
Test Description	Having many users log in at once
Steps to run test	1. Sign in with many different users
Expected Result	All users who just logged in should have a green dot displayed next to their name in the sidebar
Actual Result	Every user who just logged in has a green dot next to their name in the sidebar.

Test #	14 (Black Box Test)
Test Description	Hovering over the sidebar button should display the sidebar, including the list of users.
Steps to run test	1. Hover over the sidebar text.
Expected Result	The sidebar appears.
Actual Result	The sidebar appears.
Screenshot	 WHIMSICAL FOREST ACTIVE USERS ● Fairy Sparkles ● username ● username ● username ● username ● username ● username logout

Test #	15 (Regression Test)
Test Description	Adding background music that is played regardless of keypresses
Steps to run test	1. Launch application and make sure background music plays 2. Press keys to make sure that keypresses create sounds on top of the background music
Expected Result	Hear music from keypresses in addition to the background music
Actual Result	Hear music from keypresses in addition to the background music

Test #	16 (Black Box)

Test Description	Backend Test: Checking JDBC is connecting to the database properly
Steps to run test	1. Run application, press log-in to sign in with your google account. 2. The backend servlet will be called and the LoginServlet will run 3. LoginServlet will try to connect to database with the JDBCDriver class
Expected Result	No connection errors showing up in Console in Eclipse.
Actual Result	Nothing outputted in Console, showing no issues.
Screenshot	

```
Tomcat v9.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (Nov 4, 2018, 8:53:59 PM)
Nov 04, 2018 8:54:02 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/9.0.11
Nov 04, 2018 8:54:14 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for
Nov 04, 2018 8:54:27 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for
Nov 04, 2018 8:54:28 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Nov 04, 2018 8:54:28 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Nov 04, 2018 8:54:28 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 26160 ms
JDBC Connection established!
```

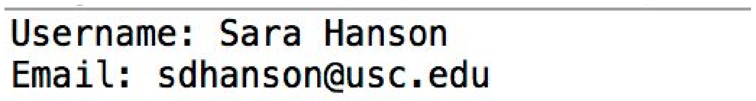
Test #	17 (Black Box)
Test Description	The waiting times in between letters in playback songs should be very similar to the waiting times in the original song recording
Steps to run test	1. Log into the application with two users 2. One of these users should have a non-empty song string associated with their name in the database
Expected Result	The playback song should match the originally recorded song
Actual Result	Same as expected result. The playback song is audibly indistinguishable with the original recording

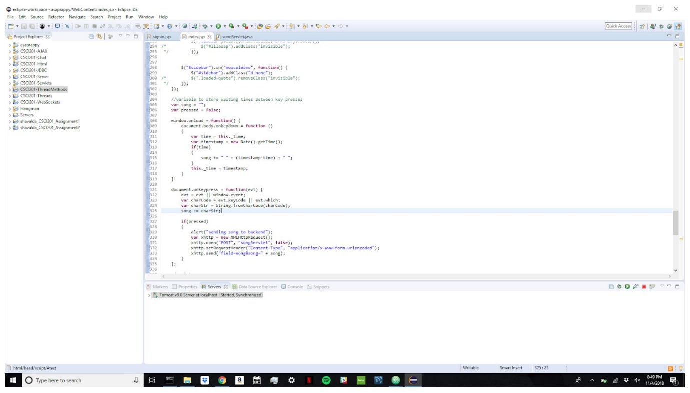
Test #	18 (White Box, Architecture Test)
Test Description	The server should be able to broadcast information to many different clients.
Steps to run test	1. Have a loop that creates many instantiates many different

	servers. 2. Trigger one client to send a message to the server. 3. Count the outgoing messages. 4. Check count with count of clients.
Expected Result	The count of outgoing messages should equal the count of the clients.
Actual Result	The number of clients was equal to the number of outgoing messages.

Test #	19 (White Box)
Test Description	Backend Test: Successfully updating the user active status in the database.
Steps to run test	1. In LoginServlet, whether the logged-in user is already in the database or not, their row in the table will have the "active" set to true by the servlet. 2. There will be a function that outputs the user's info along with their active status. Run that function.
Expected Result	(Go to the database in MySQL Workbench and run the database to check if the user that just logged in has their "active" column set to true.) Running the test above, all users should have the correct status set.
Actual Result	
Screenshot	

Test #	20 (Unit Test)
Test Description	The user login should successfully send both the user's full name and email to the LoginServlet.
Steps to run test	1. Log in. a. A function CheckLogin() will output the full name and email in the LoginServlet.
Expected Result	The backend LoginServlet in Eclipse should print out the correct full name and email of the user that's just logged in.

Actual Result	The LoginServlet printed 'Sara Hanson', sdhanson@usc.edu for the given input 'Sara Hanson', sdhanson@usc.edu
Screenshot	 ``Username: Sara Hanson`` ``Email: sdhanson@usc.edu``

Test #	21 (Unit Test)
Test Description	The user attempts to record a song without pressing the record button.
Steps to run test	1. Run the application and log in as a user 2. Type a sequence of characters into the keyboard
Expected Result	The song will not be recorded.
Actual Result	Same as expected result. Since the record button was not pressed, the song string will never be passed to the backend. There will be no update to the song attached to the user in the database.
Screenshot	 The code showing how the song recording functionality works

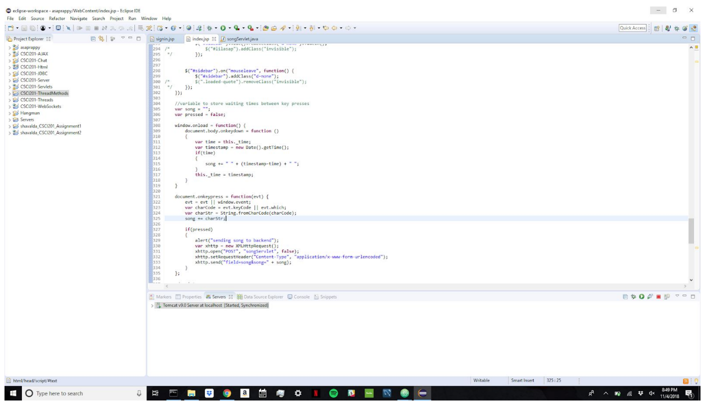
```
/*          $("#lilasap").addClass("invisible");
 */        });


        $("#sidebar").on("mouseleave", function() {
            $("#sidebar").addClass("d-none");
/*          $(".loaded-quote").removeClass("invisible");
 */        });
    });

    //variable to store waiting times between key presses
    var song = "";
    var pressed = false;

    window.onload = function() {
        document.body.onkeydown = function ()
        {
            var time = this._time;
            var timestamp = new Date().getTime();
            if(time)
            {
                song += " " + (timestamp-time) + " ";
            }
            this._time = timestamp;
        }
    }

    document.onkeypress = function(evt) {
        evt = evt || window.event;
        var charCode = evt.keyCode || evt.which;
        var charStr = String.fromCharCode(charCode);
        song += charStr;

        if(pressed)
        {
            alert("sending song to backend");
            var xhttp = new XMLHttpRequest();
            xhttp.open("POST", "songServlet", false);
            xhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
            xhttp.send("field=song&song=" + song);
        }
    };
```

Test #	22 (Unit Test)
Test Description	Previously logged in users should maintain their previous whimsical screen name after their first login
Steps to run test	1. Log in and check the whimsical username 2. Log out and log back in, check username again

Expected Result	The first username should match the second username.
Actual Result	Ben Brooks logged in with bjbrooks@usc.edu and was assigned "Magical Lemon" as a whimsical username. He logged out and clicked log in again to see that his username was still "Magical Lemon".

Test #	23 (Unit Test)
Test Description	Correctly determining whether a user that's just logged in is already recorded in the database or not.
Steps to run test	1. Run a program to continually add and extract users from the database. a. A function AddUsers() will run the SQL INSERT function to insert a user with a unique emails 'a@usc.edu', 'b@usc.edu', 'c@usc.edu', and fake usernames, 'a', 'b', 'c', etc, that are specified in a map from username to object data. b. A separate loop will use SQL SELECT to select each username and call CompareInformation(username). c. CompareInformation will query the map for the given username and compare the information in the object to the information selected from the database i. If the information does not match, the program will print "ERROR".
Expected Result	No output should be given, indicating there was not an ERROR.
Actual Result	No ERROR messages.

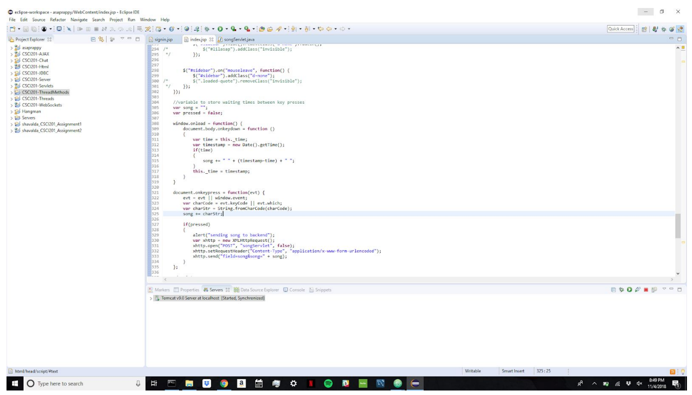
Test #	24 (Unit/Stress Test)
Test Description	The user begins to record a song and never presses the stop recording button.
Steps to run test	1. Run the application and log in as a user. 2. Press the record button, and type a sequence of characters into the keyboard. 3. Attempt to continue using the application by clicking on the sidebar.

Expected Result	The song will never be recorded.
Actual Result	Same as the expected result. The onKeyPress() function will continue running and record all of the user's key presses, regardless of whether or not they are part of an intended song string. The song string will never be sent to the backend and never updated in the database.
Screenshot	 The code showing how the song recording functionality works.

```
/*              $("#lilasap").addClass("invisible");
 */         });


        $("#sidebar").on("mouseleave", function() {
            $("#sidebar").addClass("d-none");
/*          $(".loaded-quote").removeClass("invisible");
 */     });
    });

    //variable to store waiting times between key presses
    var song = "";
    var pressed = false;

    window.onload = function() {
        document.body.onkeydown = function ()
        {
            var time = this._time;
            var timestamp = new Date().getTime();
            if(time)
            {
                song += " " + (timestamp-time) + " ";
            }
            this._time = timestamp;
        }
    }

    document.onkeypress = function(evt) {
        evt = evt || window.event;
        var charCode = evt.keyCode || evt.which;
        var charStr = String.fromCharCode(charCode);
        song += charStr;

        if(pressed)
        {
            alert("sending song to backend");
            var xhttp = new XMLHttpRequest();
            xhttp.open("POST", "songServlet", false);
            xhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
            xhttp.send("field=song&song=" + song);
        }
    };
```

Tomcat v9.0 Server at localhost [Started, Synchronized]

Test #	25 (Unit Test)
Test Description	The user begins to record a song and presses the stop recording button after finishing playing the song.
Steps to run test	1. Run the application and log in as a user. 2. Press the record button, and type a sequence of characters into the keyboard. 3. Press the stop recording button once finished.
Expected Result	The song will be correctly recorded.
Actual Result	Same as the expected result. Once the record button is pressed, the user's key presses and waiting times in between each key press will be recorded in a song string. Once the stop recording button is pressed, the string will be passed to the backend and the user's database song value will be updated.

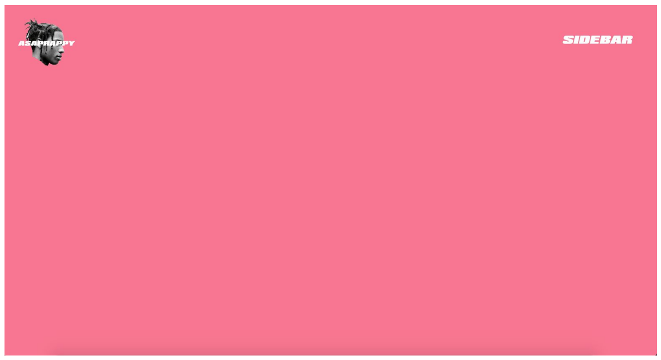
Screenshot	 The code showing how the song recording functionality works

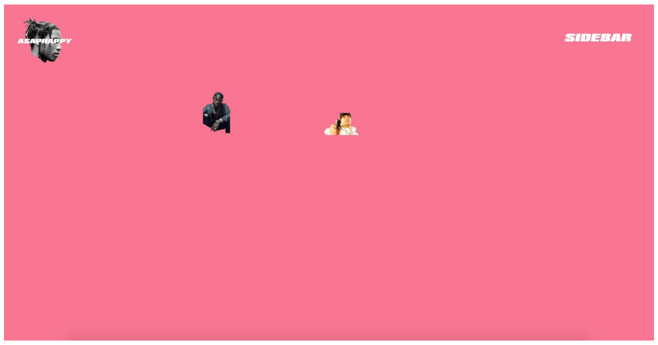
```
/*              $("#lilasap").addClass("invisible");
 */         });


        $("#sidebar").on("mouseleave", function() {
            $("#sidebar").addClass("d-none");
/*          $(".loaded-quote").removeClass("invisible");
 */     });
    });

    //variable to store waiting times between key presses
    var song = "";
    var pressed = false;

    window.onload = function() {
        document.body.onkeydown = function ()
        {
            var time = this._time;
            var timestamp = new Date().getTime();
            if(time)
            {
                song += " " + (timestamp-time) + " ";
            }
            this._time = timestamp;
        }
    }

    document.onkeypress = function(evt) {
        evt = evt || window.event;
        var charCode = evt.keyCode || evt.which;
        var charStr = String.fromCharCode(charCode);
        song += charStr;

        if(pressed)
        {
            alert("sending song to backend");
            var xhttp = new XMLHttpRequest();
            xhttp.open("POST", "songServlet", false);
            xhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
            xhttp.send("field=song&song=" + song);
        }
    };
```

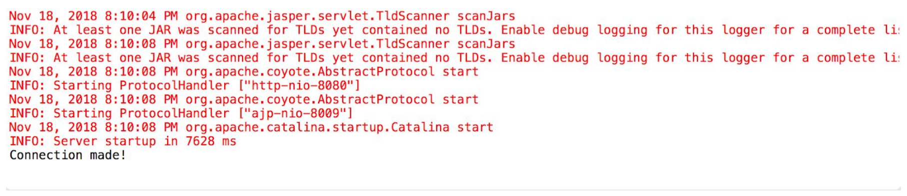
Tomcat v9.0 Server at localhost [Started, Synchronized]

Test #	26 (Black Box)
Test Description	The database is scalable and remains persistent as we add multiple users.
Steps to run test	1. Run function that inserts several hundred users into the database. 2. Read the database's user list. 3. Compare pulled list with the inserted list. 4. Automatically output differences.
Expected Result	The pulled list should exactly match inserted list.
Actual Result	Insert of ["Ben Brooks", "Shreya Havaldar", "Sara Hanson", "Nian Xu Wang", "Kevin Yeung"] exactly matches sql database output of ["Ben Brooks", "Shreya Havaldar", "Sara Hanson", "Nian Xu Wang", "Kevin Yeung"]

Test #	27 (Unit Test)
Test Description	The user requests a song from another user who doesn't have a song attribute in the database.
Steps to run test	1. Log into the application in 2 tabs. 2. Of these, one should be a new user, and the other a returning user. 3. From the returning user's page, request the song from the new user.
Expected Result	There will be a popup message that says the user does not have a

	song yet.
Actual Result	The database attribute will be an empty string, so nothing will be displayed on the landing page.
Screenshot	

Test #	28 (White Box)
Test Description	Pressing a username in the sidebar should trigger the playback functionality.
Steps to run test	1. Log in as User 1. 2. Record a song sequence. 3. Log out. 4. Log in as User 2. 5. Press User 1's username in the sidebar.
Expected Result	Multiple images and sounds should appear. Each image and sound should correspond to the sequence recorded by User 1.
Actual Result	Multiple images and sounds appeared, replicating the song sequence recorded by User 1.
Screenshot	

Test #	29 (White Box)
Test Description	Checking if websocket is connected
Steps to run test	Connect a web front end client by going to the asaprappy website with a web browser.
Expected Result	“Connection made!” will be printed in the Eclipse console
Actual Result	“Connection made!” is printed, and the client is able to access the website.
Screenshot	

```
Nov 18, 2018 8:10:04 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete li
Nov 18, 2018 8:10:08 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete li
Nov 18, 2018 8:10:08 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Nov 18, 2018 8:10:08 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Nov 18, 2018 8:10:08 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 7628 ms
Connection made!
```

Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

AsapRappy High-Level Requirements

Changes from 11/19/2018
Changes from 11/04/2018

Background

Historically, web applications for visualizing music have enjoyed prolific success. DADIM, a "trippy" music visualizer, couples songs with geometric, repetitive visuals. Patatap takes this music visualization one step further, giving users the ability to trigger sound/visual combinations through key presses. Coined a "sound and animation kit," Patatap has been featured in The Verge, Product Hunt, and Fast Company. The Rap Board, an application for playing sound bites from popular rappers, has been featured in Paste Magazine and Vulture. These applications are known for their addictive, joyful, or irreverent qualities, carving out a space for creative audio/visual endeavors in a casual manner. Our application will exist in this space, combining animation/sound kits with popular culture music to create an addictive music creation experience.

General Goal

The general goal of this application is to make an addictive, interactive, playful tool for visualizing music. A jovial aura will be achieved through simple mechanics, i.e. pressing a key to trigger an audio and visual cue and a pop-culture theme. Visually, the application should be aesthetically pleasing and easy to use. The simple aesthetics will be achieved through a one-to-two page website with bold colors, geometric shapes, and an omnipresent active users bar. Finally, the application should have a limited scope for user-to-user interaction, adding to the interactivity, while maintaining ease of use. Together, the playfulness and ease-of-use will create an addictive application for users to create, share, and visual music in a casual manner.

Core Features

The core functionality of this application will be transforming the user's keyboard into a music visualizer. Every key will correspond to a sound accompanied by a graphic on the user's screen. The graphics and sounds will also all be related to pop culture, creating a themed music experience for the user. In addition, the user will be able to see who else is using the application through the active users bar, a section at the top of the interface showing all other users currently creating music. ~~Using the active users bar, a user can request recorded music from other active users.~~ Using the user status bar, users will be able to play recorded music from other users. When not a registered user, functionality will be limited to only a few keys on the music visualizer keyboard. After creating an account, the user will be able to use every key available. ~~The user will also be able to save created music after logging in.~~ The user will be able to save music after logging in.

Stretch Goals

Increased user-to-user interaction would add to the functionality of the tool but is not required. ~~Ideally, users would be able to “share screens” and collaborate on the same screen to produce audio/visuals at the same time.~~ In addition, expanding the scope of audio/visual production would be useful. ~~Users would be able to record multiple creations, playback previously saved sequences and re-record over them.~~ Users will be able playback previously saved songs. Users will also be able to listen to other users saved songs by clicking their username in the sidebar. The application should also have at least ten audio/visual combinations, aiming for one for each letter on the keyboard. Guest accounts should be limited in their ability to access those combinations ~~as well as record their creations.~~ Users will not be able to playback or share their music sequences. ~~Users should be able to play a beat in the background as they create audio/visual combinations.~~

Technical Stacks

The backend consists of a database, a server, and the application. The database will be implemented with MySQL, the program will run on an Apache Tomcat server, and the application will be written in Java. HTML, CSS, and JavaScript will be used to design the front-end, and in addition, the libraries howler.js for audio management, tween.js for animation in general, and paper.js for image animation will be used to add functionality to the application. WebSocket technology will be used to connect the frontend web browser with the backend server.

Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

AsapRappy Technical Specifications [Changed to meet formatting requirements]

Changes from 11/18/2018
Changes from 11/04/2018

Landing Page

Front-End:

Web Interface:

The web interface will take 16 hours, including four to research the audio/visual pairs, four to set up the audio/visual combinations, four to create the sidebar, and four to integrate the interface with the backend. The styling of the homepage will use Bootstrap and CSS.

The web interface needs to have a minimalistic homepage with a sidebar including instructions and a login button. ~~The instructions will be contained in a text box in the sidebar.~~ The instructions will be in the body of the page. When a user presses WASD on the signed out page, the instructions will disappear. If the user hovers over the AsapRappy logo, the instructions will reappear. Similarly, on the logged in page, pressing any key a-z or A-Z will cause the instructions to disappear. The instructions on the logged in page will tell the user to press any button a-z, hover over the sidebar to view user activity, and use the small recording bar to record sounds. They must indicate that a logged out user has access to four audio/visual combinations triggered by pressing ASDF keys, and indicate the user can log in to access all 26 keyboard combinations.

The JavaScript section of the project will take four hours; two hours to set up the event responses for the audio/visual combinations and two hours to set up the Google API JavaScript calls for signing in and out. The web interface will include audio/visual interactions. These interactions will be programmed in JavaScript using the howler.js and paper.js libraries. Sound data will be stored in a map that maps letters to sounds. There will be 26 sounds, each stored as a howler asset. When a user presses a key, javascript will query the map by the letter pressed and use the howler play function to play the sound. At the same time, an image will be created at a random point on the screen using the math random function, and paper.js animate will be used to move the image across the screen.

Back-End:

The landing page does not require any additional backend functionality to operate properly

Logged-in Page

Front-End:

Pressing the login button will trigger the Google API default log-in pop-up, which requires a username and password field.

Upon verification, the user will be redirected to the initial homepage, but the sidebar will now include ~~instructions~~, a logout button, a whimsical screen name (different from the user's email), and a list of active users. The sidebar will be triggered by hovering over text that says "sidebar" in the upper right-hand corner of the screen. The instructions in the body of the screen will explain the user has access to 26 keyboard combinations. The list of active users will be a scrollable list of users with their status indicated by green (active) and grey (inactive) circles. The audio/visual sequence will be triggered by keypresses. Each keypress will create a pre-programmed image animation to be displayed on the "canvas", while an audio snippet is played. The canvas stretches the entire height of the screen and fills the width from the left up until the sidebar. The audio/visual pairs will be pre-programmed to last for as long as the audio snippet lasts with the max visual lasting four seconds. Multiple audio/visual pairs can be triggered at once by pressing keys simultaneously; howler.js manages multi-audio clips. Upon logging out, the homepage will return to the previous logout page.

Furthermore, JavaScript will be used to trigger the log in/log out functionalities. When the user presses the log in button, a JavaScript function will retrieve the current state of the user from the Google API.

~~If the user is logged in, an Ajax call will be made to the servlet in order to add user information to the database and reload the logged in homepage. Similarly, if the user is not logged in, an Ajax call will be made in order to reload the logged out homepage.The Ajax section time approximations are included in the JavaScript and web interface estimates.~~

~~An Ajax call will be used to dynamically display the changed user interface upon log in/ log out. Once the user logs into the application with their Google account, the text on the sidebar will change from an introductory paragraph explaining the purpose of the application and inviting users to sign in to a paragraph welcoming users to the full functionality of the application.~~

~~In addition, an Ajax call will be used to display the users who are active/inactive. All the users who have logged into the application before will be included, and those online will be separated into an "active now" section. Upon clicking the "show friends" button, the user will be able to see who is active now and who is not. The need for an Ajax call, in this case, will be to enable only the sidebar to refresh so the user does not lose the music they have created in the homepage.~~ Upon logging in, the user will be redirected to a different page called index.jsp. From here, the user will have access to all 26 key combinations and be able to view/interact with other active users on the application.

Back-End:

Multithreading:

We expect to spend approximately three hours on the multithreading component. For multithreading, we simultaneously display visual animations on the home page, while playing music. By prompting the JVM to time slice the visual and aural threads, we will achieve the simultaneous playback of the visualization and sound. Additionally, our project will incorporate the functionality to record sequences of key presses for future playback. This will utilize multithreading by having a visual and audio playback for the user, while also simultaneously storing the user's keypresses into a database. Lastly, our project will have a feature that shows which other users are online. A thread will constantly poll the database to see which users are active, while the main thread will continue to respond to keyboard input.

Networking:

We expect to spend approximately two hours on the networking component. It will involve displaying active users, requesting music samples from other users, and playing their samples. Above in the multithreading section, we talked about polling the database to check who's online. The networking aspect is that every user will be able to see from their screen which other users are active. User A can also search and click on another user B and send an request to ask for their music record. ~~If the request is approved by B,~~ B's music sample will be retrieved from the database and made available to A. User A can then play B's music sample. This was implemented using a WebSocket. We had a vector to store all active client sessions, one for every user logged into the application at a given time. The socket class will handle all of the networking components - whenever a user logs in or logs out, their sidebar status will be updated accordingly.

Database:

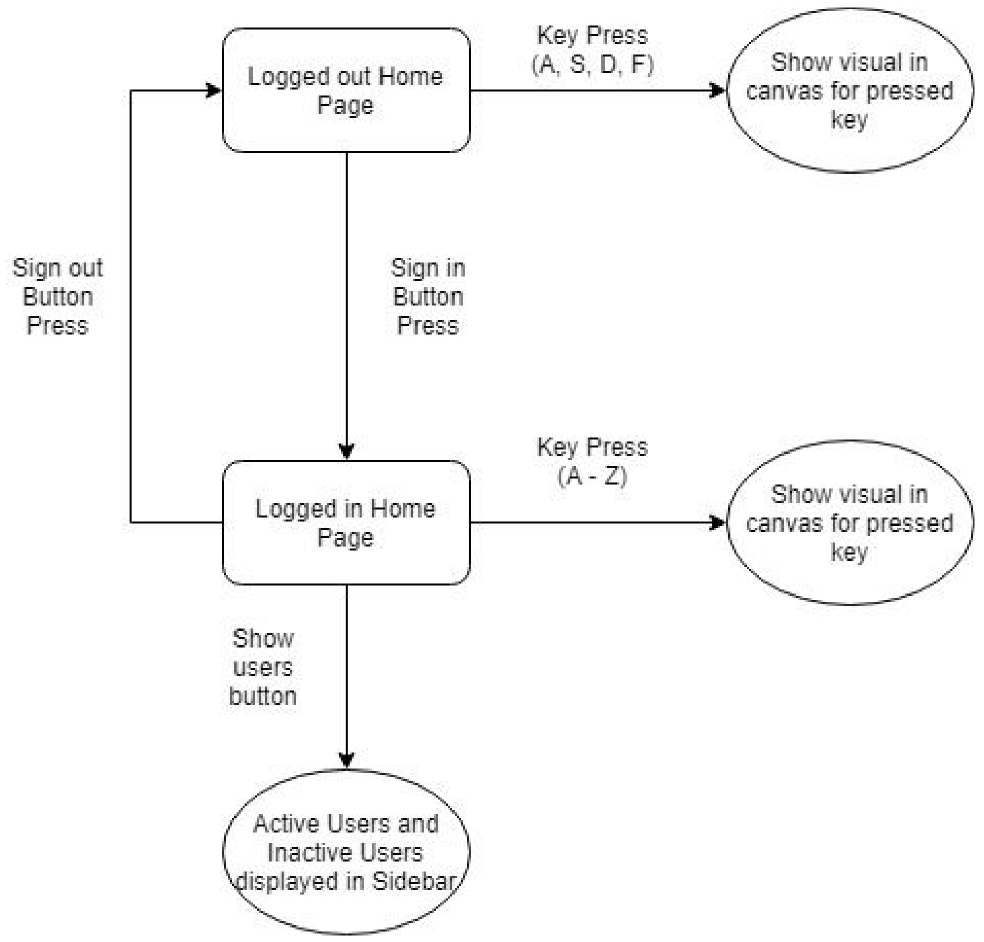
We expect to spend approximately two hours on the database component. The database will be operated through MySQL and will include an Information and Recording table. We will use the JDBC Driver to easily connect to the database from our server socket. The Information table will include an auto-incremented User ID primary key as well as each user's name, login information, and online status. The Recording table will store (at maximum) one, ten second recording that is attached to a User ID foreign key. This "recording" will actually just be a sequence of chars that represent a certain sampling per second. "Rests" will be represented by dashes while notes/button presses will be written down as the char that is selected. These recordings will be pulled from the database when we go to playback an individual's recording or when we choose to send recordings from user to user.

~~Servlet:~~

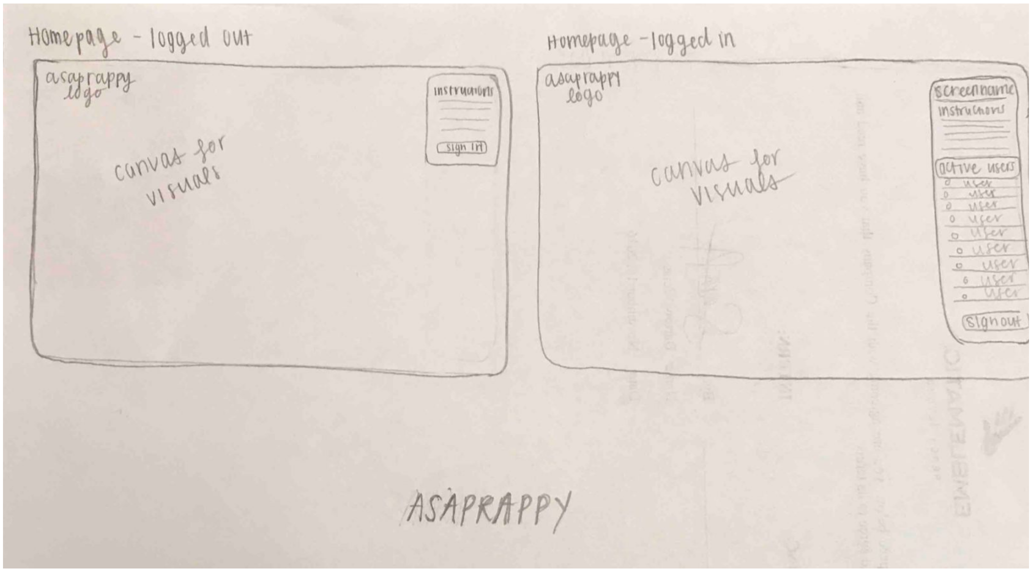
~~We expect to spend approximately three hours building the Java Servlet. The servlet will be our primary mode of communication with the database via JDBC. Specifically, it will be used to upload new user information during registration, validate user information via during login, and poll the database to check the online status of users. These interactions will be combined with the aforementioned AJAX calls. In addition, clicking record on the web application will~~

~~trigger a servlet function to begin appending to a string that will eventually be pushed to the database as a recording.~~
We do not require servlets anymore because we switched to using websockets to connect our front and backend.

Flowchart



User Interface:



Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

AsapRappy Detailed Design Document

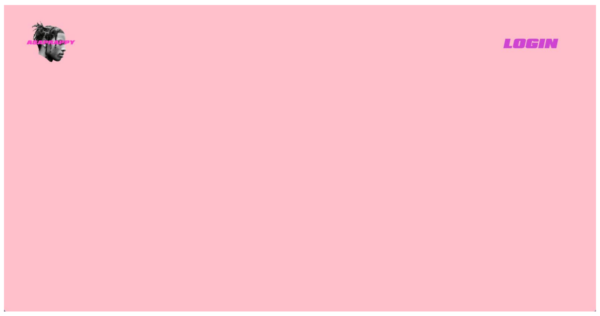
Changes from 11/19/2018
Changes from 11/04/2018

System Requirements

Asaprappy will require a host computer/server to handle the web application's traffic and requests. Given that fairly traffic is expected to be fairly limited, a group member's computer can simply run the server through Eclipse. Any connected computer will require internet access to communicate with the server. Clients will also require a web browser, preferably Google Chrome, on their computer in order to access and run the web application.

Landing Page

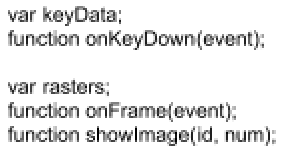
Asaprappy has two main pages, the landing page and the logged in page. When users first navigate to the site, they will see the landing page. This page is the only page available to users who are not logged in, and it has limited site functionality. A GUI mockup of the landing page is as follows:



The GUI has a logo in the upper-left corner and a login button in the upper-right corner. The entire page is a canvas for displaying audio/visual combinations. With this minimalistic mockup, client-side functionality for the landing page is limited to accessing a suite of four audio/visual combinations, viewing instructions, logging in.

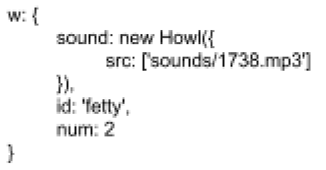
On the landing page, users can hover over the upper-left corner logo in order to view instructions. The instructions will appear in a text box, and they will ask users to press one of the four keys, ~~'A', 'S', 'D', 'F'~~, 'W', 'A', 'S', 'D' and log-in in order to access more audio/visual combinations.

On the landing page, users can trigger audio/visual combinations by pressing one of the keys: ~~'A', 'S', 'D', 'F'~~, 'W', 'A', 'S', 'D' . A script, written with Paperscript, a Javascript library for displaying images, and Howler, a Javascript library for playing audio, manages the audio/visual combination functionality on the client side. The script contains the following functions and global variables:



```
var keyData;
function onKeyDown(event);

var rasters;
function onFrame(event);
function showImage(id, num);
```

The keyData global variable maps keys to key data objects.



```
w: {
        sound: new Howl({
                src: ['sounds/1738.mp3']
        }),
        id: 'fetty',
        num: 2
}
```

In this example, the key 'W' is mapped to a key data object. The key data object has three variables: sound, id, and num. Sound is a Howler sound object that holds an mp3 audio clip. The id variable specifies the id name of an HTML image that is hidden in the canvas body. The num variable specifies the animation path number, which specifies how the rapper image should traverse the canvas. The 'canvas' is the entire screen.

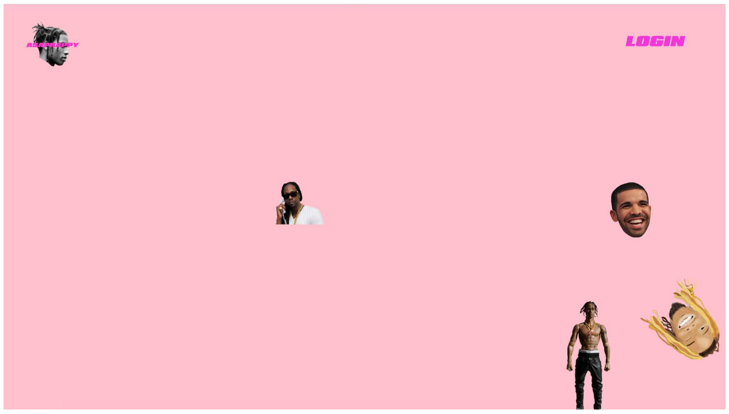
Paperscript provides two functions: onKeyDown(event) and onFrame(event).

The onKeyDown(event) function is triggered when a user presses a key. The event variable specifies which key was pressed. If the key exists in the keyData map, the function calls showImage(id, num) and Howler's play() function, which plays the keyData sound specified by the sound variable for the duration of the sound clip.

The showImage(id, num) function loads the image with the specified id at a random point on the canvas and adds an object with the image and given path number to the global rasters array.

The onFrame(event) function is called every frame by Paperscript. The function iterates through the rasters array and updates each image's position based on its animation path. For example, num: 2 corresponds to an animation path that increases the image scale and moves it to the left every frame. Paperscript animation system gives access to '.position.x' and '.position.y' of a function making animation simple.

The four images available on the landing page can be seen in the following mockup:



~~Clicking on the login button will prompt an onSignIn() javascript function that will direct to a Java Servlet. The servlet will authenticate the user and check to see if they have logged in before. If not, a "whimsical screen name" or screen username will be generated for them and they will be attached to a new entry in the user database via email. The user is then redirected to the logged in page via an onreadystatechange function.~~

Clicking the login button will prompt the Google API. The Google API will require a user to log-in through an email. Once the email is authenticated, the user will be redirected to the index page. The username and email will be passed through the URL.

Server-side

The landing page requires no server-side functionality because the landing page is for guest users, and guest users do not have access to the database or other more advanced features.

~~**Landing Servlet**~~

~~Getting the information on the user that's trying to log in: (Through the service() function)~~
~~The onSignIn function will pass the user's unique email and full name to the servlet. Using the JDBC Driver Class, we will check this information against currently stored users in the database. If the user does not exist within the database, their information will be saved and assignScreenName() will be called.~~

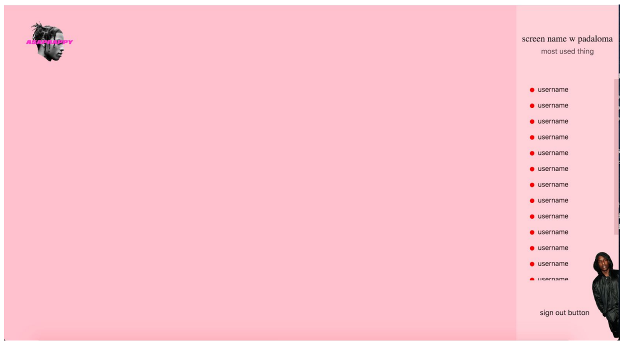
~~Giving the user a whimsical screen name:~~
~~There will be a hard coded static array of whimsical screen names in this servlet. The function assignScreenName() will pick a random name from that array and assign it to this user, and store it in the database record for this user.~~

~~**JDBC Driver Class**~~

~~The purpose of this class is to provide a convenient way of using JDBC to access the database from the backend server code.~~
~~Connecting to the database:~~
~~In this JDBC Class, there will be a connect() function where it will start a connection to the MySQL database. The database will be hosted on the host computer/server, which is already known, and so the address of the server and the database's username and password will be hard coded into this class.~~
~~Disconnecting from the database:~~
~~At the end, when the user has finished using the application, the close() function will be called before the user is logged out to close any connections to the database.~~

Logged In Page

When users have successfully logged in, they will have access to the logged in page. This page has full site functionality, including access to the 26 audio/visual combinations. A GUI mockup of the landing page is as follows:



WebSockets

When the index page is loaded, it will initialize a WebSocket and send the username and email to the ServerSocket. The server will query the database for the user information. If the user is a returning user, their screen name will be retrieved from the database, along with their song. Otherwise, a screen name will be generated and stored for a new user. The screen name and song will be sent back to the index page.

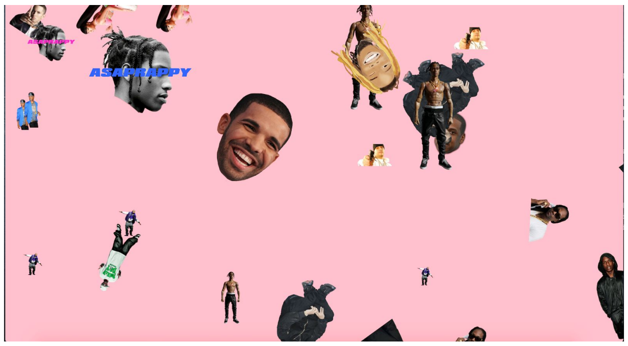
When the message is received on the front-end, the screen name will update.

When a user hovers over the image in the lower right-hand corner, a JQuery listener will trigger the sidebar to appear. The username will appear at the top of the sidebar, ~~along with instructions~~, ~~and the name of the "most pressed" rapper key~~. Furthermore, when the user hovers over the sidebar, the WebSocket will make a call to the server, retrieving the status of all users along with their songs. Thus, a list of other users, with circles indicating the status of each user, will appear. Finally, a sign out button will be provided at the end of the sidebar.

Display Design

Instructions will appear in the body of the homepage. After pressing a key, the instructions will disappear. A JavaScript function will change the instructions display to 'd-none' when a key is pressed. If the user hovers over the logo in the upper left-hand corner, the 'd-none' class will be removed from the instructions. If the user stops hovering over the logo, a JavaScript function will change the instructions display to 'd-none' using JavaScript's built-in 'onmouseexit' event.

The client-side script for managing the audio/visual combinations is the same script as the one on the landing page, but with keyData for all 26 keys. An example of these images is seen in the following mockup:



The sidebar functionality will be implemented with JavaScript, AJAX, and JQuery.

In order to display the sidebar, users will hover their mouse over the Asap Rocky image in the lower-left corner. The "onhover" functionality will be implemented with JQuery's .on("mouseenter") and .on("mouseexit") functions. Entering the image area will remove the sidebar's "d-none" class, rendering it visible. Exiting the sidebar area will add the "d-none" class to the sidebar.

The "whimsical screen name" or sidebar username will be displayed at the top of the sidebar after it is pulled from the user's entry in the database and passed to the front-end.

~~The most pressed key and status of other users will require communication between the client and the server via AJAX.~~

~~Using JavaScript's setTimeout(function, time) function, we will run the JavaScript function getActiveUsers() every 5 seconds and the JavaScript function getMostPressedKey() every 5 seconds. These will both be displayed in the sidebar.~~

~~Similarly, getMostPressedKey() will make an asynchronous call to the server and display the returned HTML in the span for the most pressed key.~~

Song Recording

The song recording functionality will be implemented as follows:
Songs will be stored as Strings the database, as attributes of the user who made them. Each song string will have 2 components - the keys themselves, and the waiting time between each key. A song string will look like this:

"A 1123 B 890 C 56 C 58 E"

The letters are the keys pressed by the user, and the numbers are the waiting times (in milliseconds) that pass between two key presses.

The letter keys will be stored in a String in the Paperscript part of the index.jsp file - the onKeyDown(event) function will trigger the audio/visual combination attached to the corresponding key as well as append the key to the song string. The waiting times will be stored in a string as well (separated by spaces) and retrieved using a javascript function in the jsp file. ~~Both strings (letters and waiting times) will be sent to the song storage servlet where they will be processed and sent to the database as a song attribute for a user.~~ Strings will be sent to the ServerSocket through a WebSocket when the user presses the stop recording button.

Song Playback

When a user presses the play button to playback their previously recorded song or presses another user's screen name, a .on('click') event will call the song playback function, playback(song) will be called. This function takes in a song string. It instantiates a WebWorker. This WebWorker runs the script playback.js.

Playback.js parses the song string and iterates through it. When it encounters a letter, it sends the letter back to the index page. The index page then plays the sound associated with the letter. When the Playback.js function encounters a number, it waits for the specified amount of milliseconds before resuming parsing the string, ensuring the correct pauses in the song sequence.

Server Side

JDBC Driver Class

The purpose of this class is to provide a convenient way of using JDBC to access the database from the backend server code.
Connecting to the database:

In this JDBC Class, there will be a connect() function where it will start a connection to the MySQL database. The database will be hosted on the host computer/server, which is already known, and so the address of the server and the database's username and password will be hard coded into this class.
Disconnecting from the database:
At the end, when the user has finished using the application, the close() function will be called before the user is logged out to close any connections to the database.

WebSocket
The Websocket is used to facilitate communication between the Client and the Database. It uses a Vector of Session, String pairs to store and identify the various sessions that are initialized at each login. It uses the onmessage() function to enable communication between the client and the Websocket.
Log-In: When a user logs in, a session is initialized and added to the session vector. Given the user's email, the websocket will check to see if the user exists in the database. If they do, their screen name is returned back to the client. Otherwise, a screen name is generated, saved to the database and sent back to the client. The user is marked as "active" in the database and this information is broadcast to every active user. The sidebar of the user is updated to reflect the status of every user.

Log-Out: The user is marked as inactive in the database and this information is broadcast to every active user. (Their sidebars are updated to reflect the change)

Song-Recording: After a song is finished recording, the string representation is sent to the Websocket to be saved in the database (for the corresponding user)

Song-Request: The Websocket is given a request and a receive email. The request user is probed in the database to find their "song". This song is then sent to the receive user (by checking the vector of pairs) and subsequently played by the client.

~~Song Storage Servlet~~
~~Merging the song strings~~
~~The two strings (letters and waiting times) will be passed into a mergeStrings() function, where each pair of letters will be separated by a waiting time~~
~~The merged song string will be directly passed into the backend~~

~~Sending the song string to the servlet~~

~~In the service() function, using the JDBC Driver class, we will update the user's song attribute to the merged string of letters and waiting times. New songs will override pre existing ones.~~

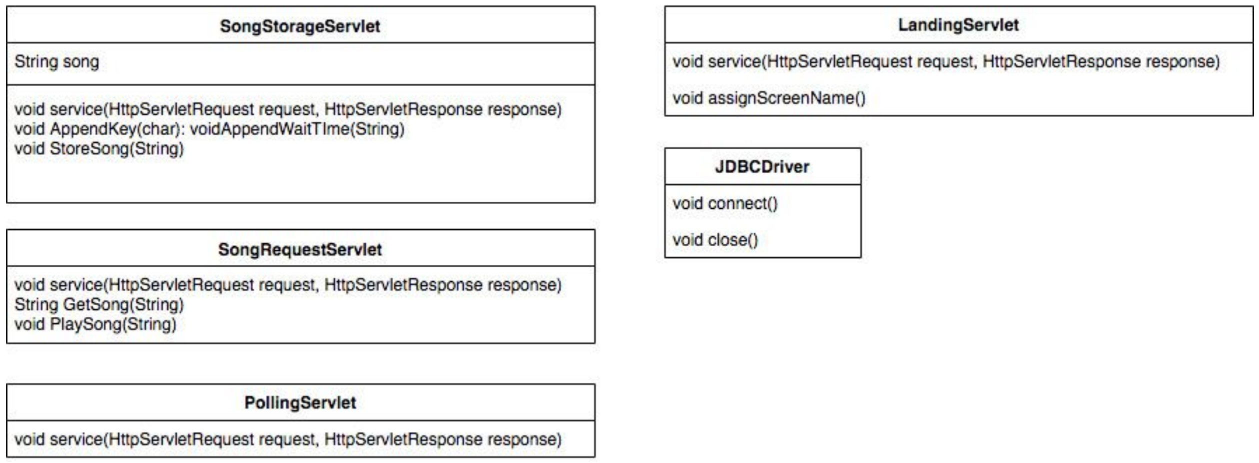
~~Song Request Servlet~~
~~Receiving the request~~
~~In the service() function, we will use the JDBC Driver class to access the song associated with the user email sent to the servlet.~~

~~Playing songs~~
~~The playsong() function will be called, triggering the associated audio/visual combination on the user's screen. We will parse the string and start a thread for every letter encountered. We will wait the designated time before starting the thread for the next letter.~~

~~Online Status Polling Servlet~~
~~Refreshing the Sidebar~~
~~We will do this by making an AJAX call from the index.jsp sidebar section every 5 seconds.~~

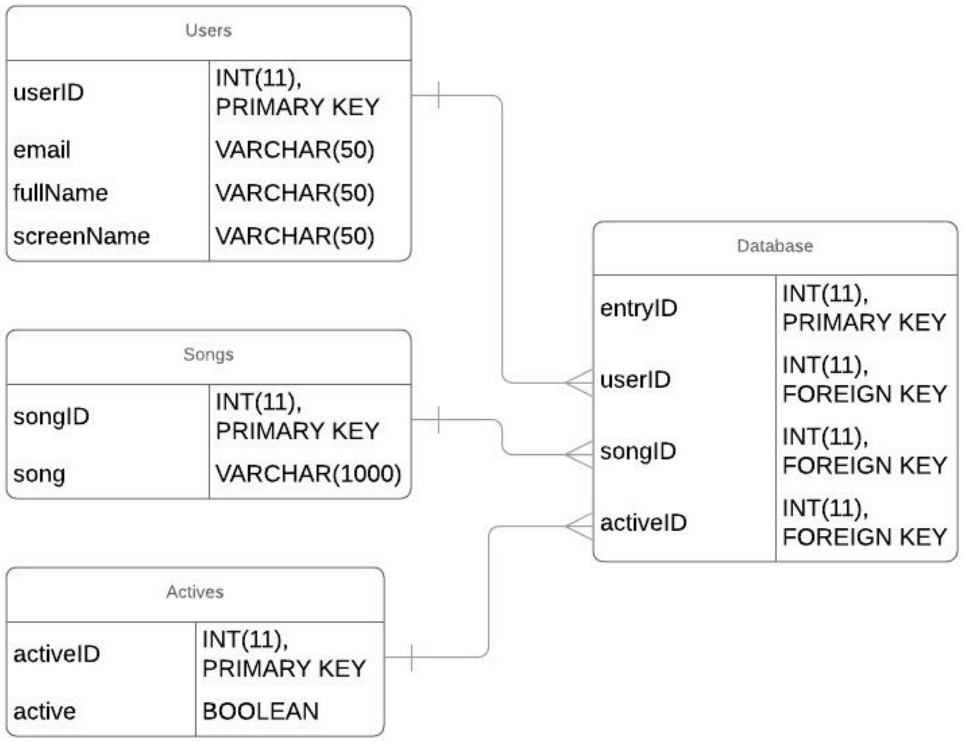
~~Polling the Database~~
~~The service() function will be called every time the ajax call is refreshed. One thread will constantly poll the database, checking the actives table, in order to properly display the current active users.~~

Class Diagram


SongStorageServlet
String song
void service(HttpServletRequest request, HttpServletResponse response)
void AppendKey(char): voidAppendWaitTIme(String)
void StoreSong(String)
SongRequestServlet
void service(HttpServletRequest request, HttpServletResponse response)
String GetSong(String)
void PlaySong(String)
PollingServlet
void service(HttpServletRequest request, HttpServletResponse response)
LandingServlet
void service(HttpServletRequest request, HttpServletResponse response)
void assignScreenName()
JDBCDriver
void connect()
void close()

Sara Hanson, Ben Brooks, Shreya Havaldar,
Nian Xu Wang, Kevin Yeung

Asaprappy Database ER Diagram


Users
userID
INT(11), PRIMARY KEY
email
VARCHAR(50)
fullName
VARCHAR(50)
screenName
VARCHAR(50)
Songs
songID
INT(11), PRIMARY KEY
song
VARCHAR(1000)
Actives
activeID
INT(11), PRIMARY KEY
active
BOOLEAN
Database
entryID
INT(11), PRIMARY KEY
userID
INT(11), FOREIGN KEY
songID
INT(11), FOREIGN KEY
activeID
INT(11), FOREIGN KEY