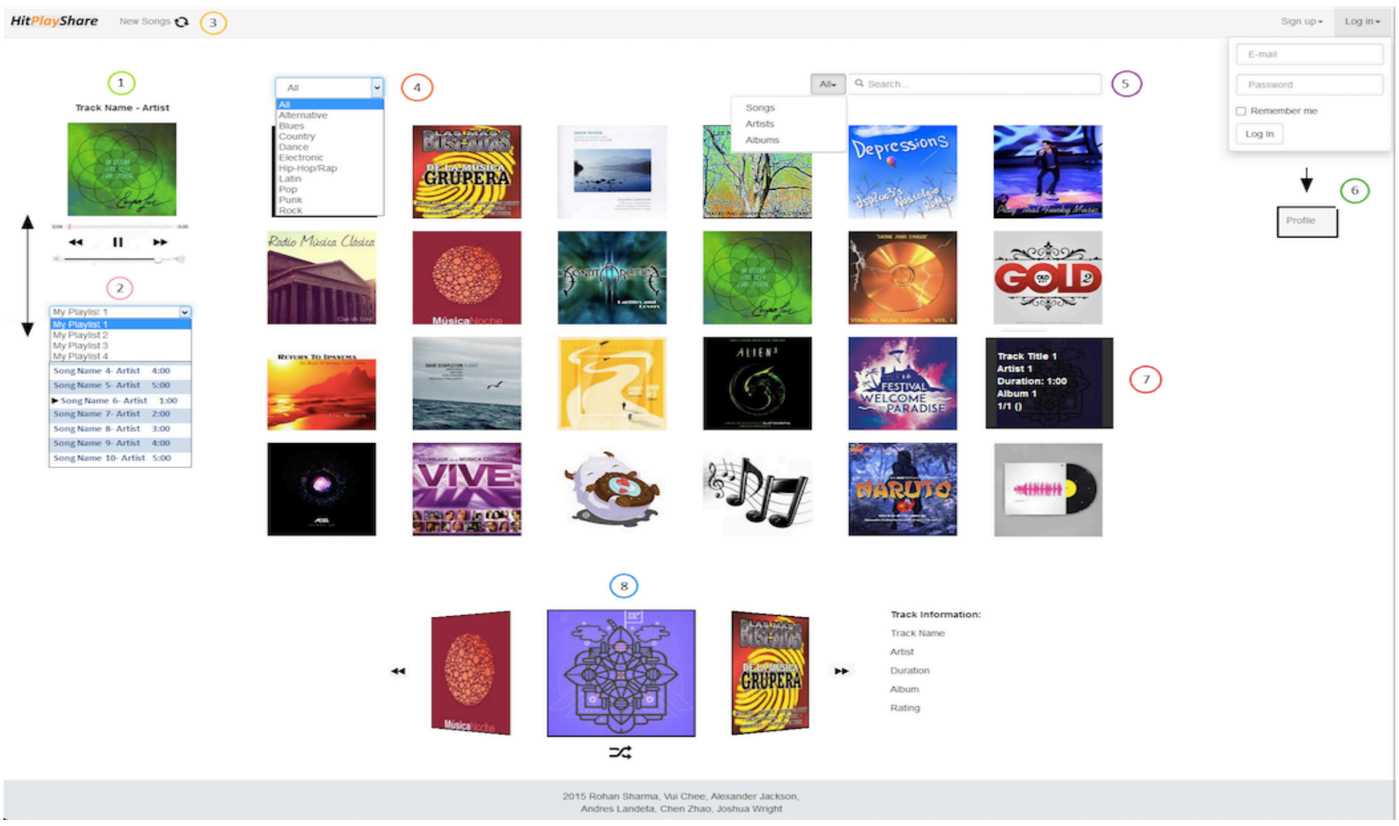
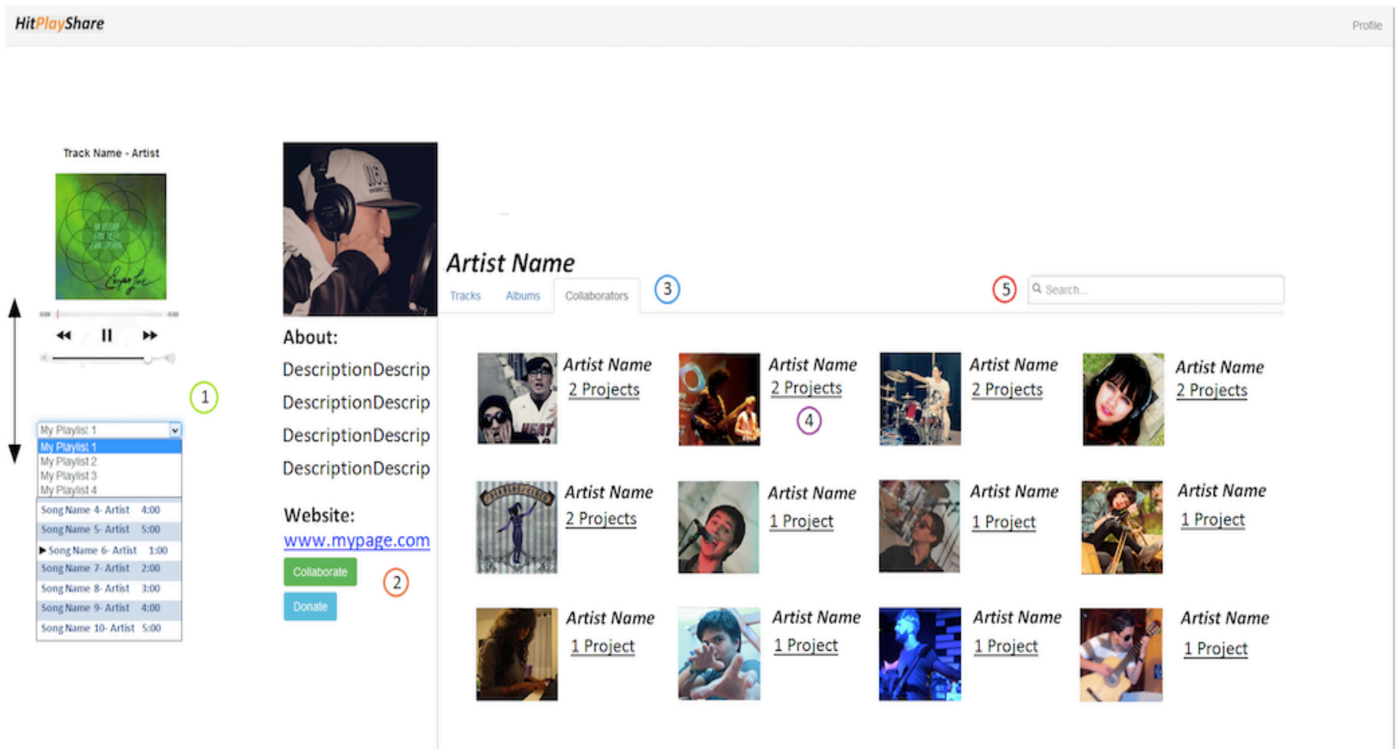
Part C : Project Plan and Application Design

User Interface Design

Home Page



User Profile Page

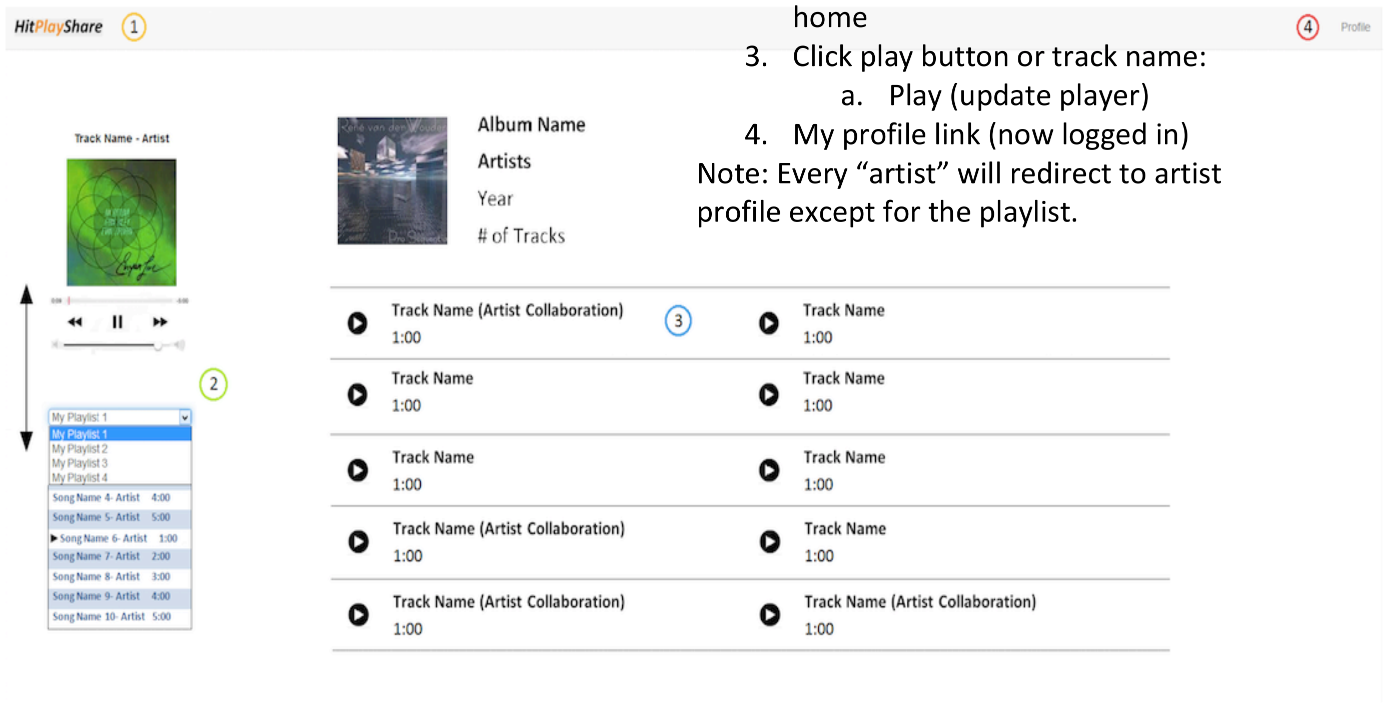


Album Page

Album Page

1. Redirect to home
2. Same usage and functionalities as home
3. Click play button or track name:
   a. Play (update player)
4. My profile link (now logged in)

Note: Every "artist" will redirect to artist profile except for the playlist.



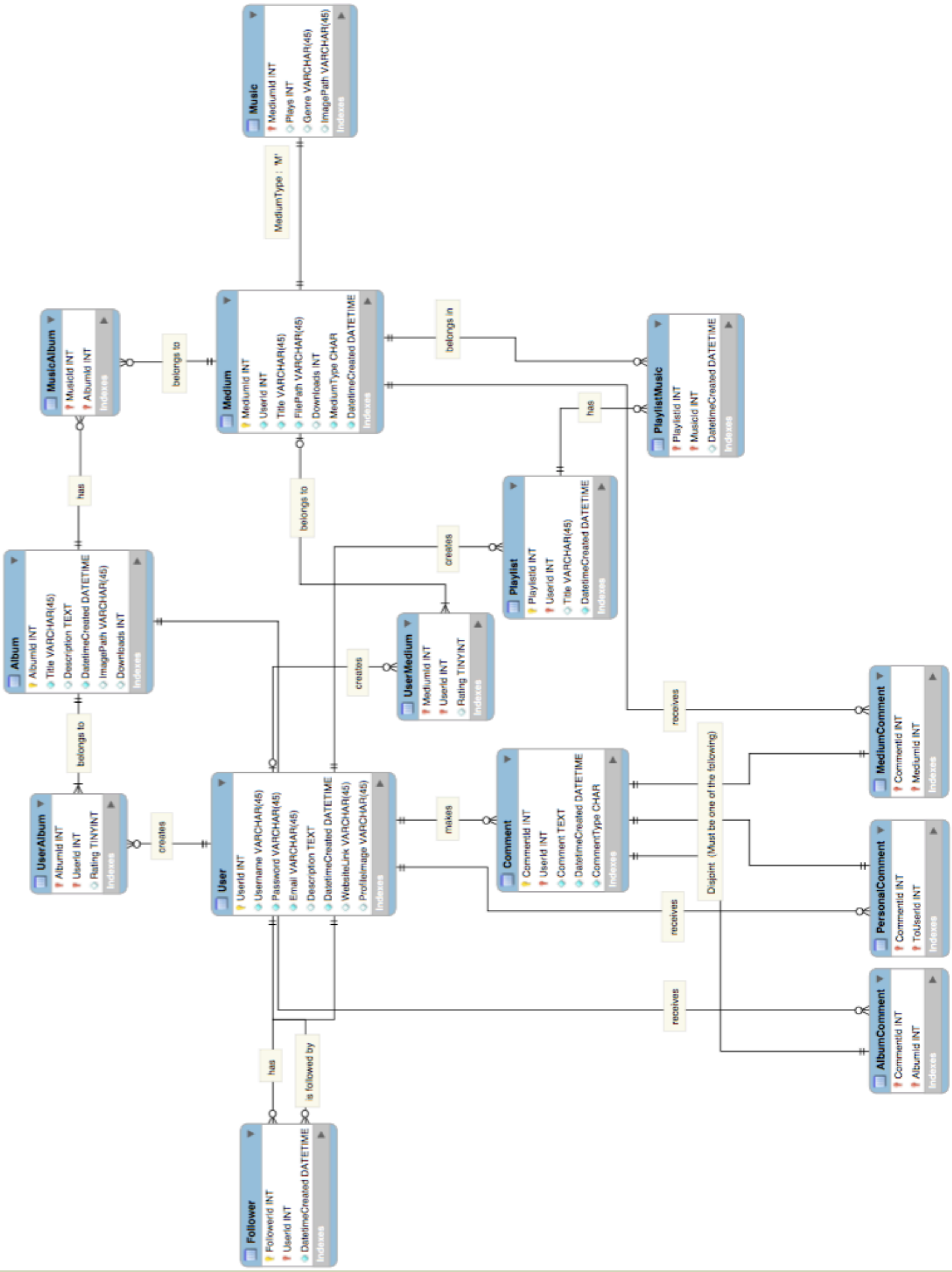
Home Page

1. Current track playing with sound control
2. Playlist Menu:
   a. Display when the user is logged in
   b. Position will go down with the page
   c. Change what the playlist will be displayed
   d. Current track playing is highlighted
   e. Tracks order can be modified
   f. Tracks can be deleted from playlist
3. Reload new tracks for the grind
4. Select track's genre (will update the grind)
5. Search bar:
   a. Search for tracks, artists, albums or all of them
   b. Will update the grind with results
6. Log in form will change to link to profile
7. Grind element:
   a. Hover:
      i. Show information
   b. Click:
      i. Track -> play and update the carousel
      ii. Artist -> redirect to profile
      iii. Album -> Redirect to page
8. Carousel with Random Tracks:
   a. A track will be added when clicked on grind
   b. You can return to previous tracks that you played
   c. Generate a random track to play
   d. Information on the right will update

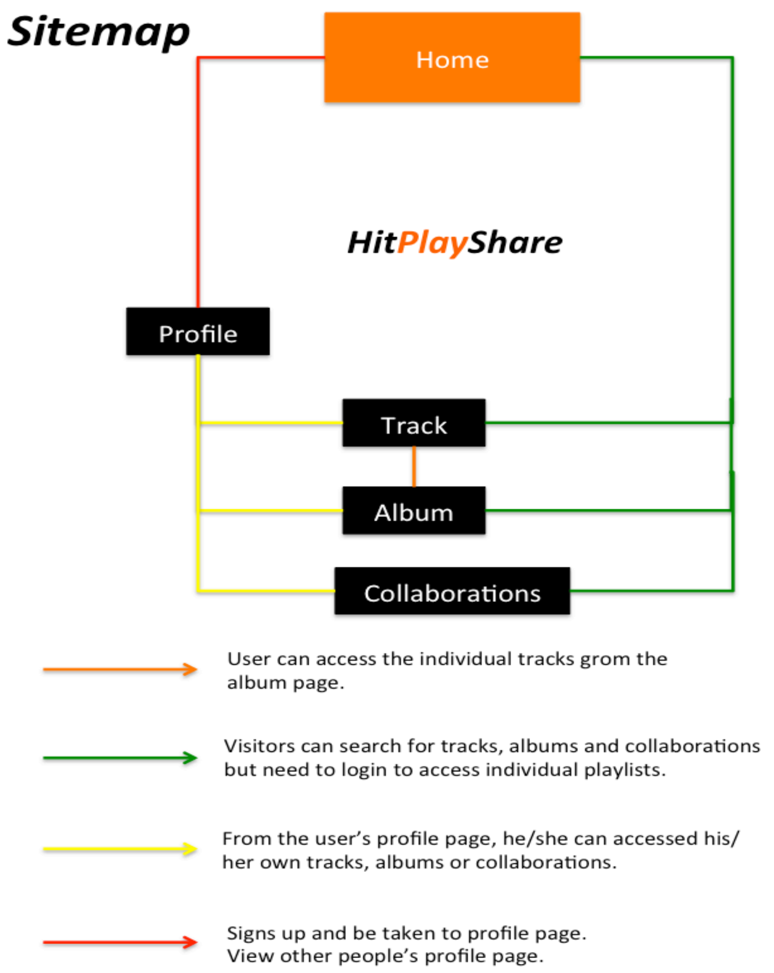
User Profile Page

1. Same usage and functionalities as home
2. Donation and Collaboration:
   a. Donation: opens a new tab with donation system
   b. Collaboration: send a request to the artist (only appear when you log in)
3. Tabs:
   a. Tracks: A grind (See profile editor for example)
   b. Albums: Almost identical to tracks (change information)
   c. Collaboration: All artists collaborated with
4. Projects: if "projects" is clicked the tab will be modified to show albums and tracks
5. Search bar: will search for matches in the current tab

Data Design


Music
MediumId INT
Plays INT
Genre VARCHAR(45)
ImagePath VARCHAR(45)
Indexes
MediumType : 'M'
MusicAlbum
MusicId INT
AlbumId INT
Indexes
belongs to
Medium
MediumId INT
UserId INT
Title VARCHAR(45)
FilePath VARCHAR(45)
Downloads INT
MediumType CHAR
DatetimeCreated DATETIME
Indexes
belongs in
PlaylistMusic
PlaylistId INT
MusicId INT
DatetimeCreated DATETIME
Indexes
has
has
belongs to
Album
AlbumId INT
Title VARCHAR(45)
Description TEXT
DatetimeCreated DATETIME
ImagePath VARCHAR(45)
Downloads INT
Indexes
creates
Playlist
PlaylistId INT
UserId INT
Title VARCHAR(45)
DatetimeCreated DATETIME
Indexes
creates
UserMedium
MediumId INT
UserId INT
Rating TINYINT
Indexes
belongs to
receives
UserAlbum
AlbumId INT
UserId INT
Rating TINYINT
Indexes
creates
User
UserId INT
Username VARCHAR(45)
Password VARCHAR(45)
Email VARCHAR(45)
Description TEXT
DatetimeCreated DATETIME
WebsiteLink VARCHAR(45)
ProfileImage VARCHAR(45)
Indexes
makes
Comment
CommentId INT
UserId INT
Comment TEXT
DatetimeCreated DATETIME
CommentType CHAR
Indexes
Disjoint (Must be one of the following)
MediumComment
CommentId INT
MediumId INT
Indexes
receives
PersonalComment
CommentId INT
ToUserId INT
Indexes
receives
AlbumComment
CommentId INT
AlbumId INT
Indexes
has
is followed by
Follower
FollowerId INT
UserId INT
DatetimeCreated DATETIME
Indexes

Navigation Design



User can access the individual tracks grom the album page.

Visitors can search for tracks, albums and collaborations but need to login to access individual playlists.

From the user’s profile page, he/she can accessed his/her own tracks, albums or collaborations.

Signs up and be taken to profile page.
View other people’s profile page.

Technology and Infrastructure

As previously shown, we have used a sitemap and a basic user interface design to demonstrate basic workings of our planned website. Now, we are going to talk about the Rails framework and architecture we will be using in construction of our web application. In addition to using Ruby on Rails, we will be using JavaScript (jQuery and plug-ins) and CSS for designing the eye-pleasing pages as well as built-in interactivity to our pages. Moreover, most user activity will stay on the same page thanks to Ajax.

First off, I am going to give an overview of a Rails application when a user types “**rails new app_name**” and creates a new Rails application. This will generate a bunch of files in a directory under the specified application name.

Overview of File Structure of Rails Application

App_name (main directory/ ones bolded indicated most important aspects)

1. **app (contains assets folder -> stylesheets, JavaScript, images)**
2. bin
3. **config (for the report, only routes.db is discussed)**
4. **db (migrations)**
5. lib
6. log
7. public
8. test (for testing models)
9. tmp
10. vendor
11. other files (Gems etc.)

Under the app folder inside your application directory, there are 6 folders: assets, controllers, helpers, mailers, models and views. And in this folder, most of your code is written here. For front–end code such as Javascript and CSS, they can be written separately and stored inside the assets directory. For ruby code, most of the code will be written in models, controllers and views. Obviously, the Rails framework adopts the model-view-controller (MVC) approach by allowing users to separate tasks under this 3 broad categories. Another important folder is the config folder which stores files that allows users to route different pages together using ruby code.

Database Management

For database management, like performing CRUD on the selected database, this is done primarily through the db folder, which houses the migrations folder. In Rails, actions on the database can be performed using "migrations". For instance, if the user wants to add a column to a table, he can alter the migration file and call "rake db:migrate" to update the database with the new migration file. Through migrations, the user can avoid having to write nasty SQL statements to change or modify certain parts of the database.

One particular thing about rails, if you want to perform queries on the database, you can simply call "rails console" on your command prompt and it will give you a console where you can query the database using object oriented language. So if you want to check a record in the User table, you can just type : "User.find(1)" , to find a record where the user id is 1.

Models

Another important thing about rails is that tables are called models (ruby classes). To create a model in Rails, the user can call "rails g model **model_name** [field_name:field_type]" to construct a class in the models folder and a migration file which specifies how the table should be created in the database. As mentioned previously, any desired modification to the table can be done through the migration file.
For relationships between entities in database modeling, they are done through the models in Rails, where special ruby syntax is used to specify many-to-many or other types of relationships.

To view all completed models after migrating, the db folder stores the schema.rb file which displays all ruby classes that were generated so that the user can keep track of all migrations regarding built tables.

Views and Controllers

The most important part of the Rails application is the controller folder, which contains the controller files which interacts with models and views of the application.
To understand how the controller works, we must also understand how html files are stored in Rails. In a Rails application, html code are stored in separate folders under a **views** folders. The folders named here are related in name to ruby files stored in the **controller** folder.
Eg. A controller file named media_controller.rb has all the actions required by all html pages stored in the media folder (under views folder).

Controllers

- **media_controller.rb**

Views

- **media**
  - **index.html.erb**
  - **new.html.erb**

Each html form inside Rails must correspond to a method inside the controller. So for every file inside the media folder, the prefix matches index and new methods located inside media_controller.rb.

So how do they interact? For example, if you want to submit a form from a html page, the values from the form are passed to the controller method (called an action in Rails) specified by the route inside routes.rb in the config folder. The method processes the values from the form and redirects the user accordingly. Since it is a class method, it is up to the creator to modify what it does.

When a controller needs to interact with a model, a method inside the controller file creates an object of that model class which is inside the models folder. Models basically serve as classes which are used inside controllers.

For html.erb files that requires JavaScript files, the <%= javascript_include_tag "some_file.js" %> tag includes the appropriate JavaScript file from assets directory without needing the user to specify its path. Similarly for CSS files, the <%= stylesheet_link_tag "some_file.css" %> does the same.

The clever use of routes and controller methods allows information to be passed from model to controller and to views and vice-versa.

Security Design
Technology ***Based Risks and*** *Design Choices*

SQL Injection
Any system that has a user interfacing with a SQL database in any way (although in particular through forms) can potentially be vulnerable to SQL injection attacks from malicious sources (or even accidentally). Although one of the most common types of attacks, SQL injection is very easily mitigated using rigorous input sanitization and strictly limiting user database access to predefined queries hardcoded into the site backend itself.
Connection Security
These days it is assumed that a site will support HTTPS connections to and from its servers, ours will be no different with SSL used to secure communications between the server and the end user to ensure nobody in between listens in on anything they shouldn't.
Direct Linking to files
We will not allow direct linking to media files on our servers via external links, there are a few simple reasons for this. First and foremost being rights management for our contributing users, depending on the privacy settings chosen for a given uploaded media the user may not desire that it be publically available, or if they do want it publically available, they may not wish for the source file to be *freely* available. Conversely, allowing external sources to link directly to our files could allow our files to be embedded in external sites, which would result in a loss of traffic, but an increase in data usage by the server, which is understandably something we want to avoid.
Backups
This should go without saying but frequent backups of all systems will be made automatically. Full backups on a weekly basis with incremental backups performed daily. Thus in the event of any catastrophic disaster, the site and its data can be saved and restored in a timely manner.

User Level Security

Access Controls
As with any web service featuring social aspects, our site will implement the ability for users to control the visibility of various aspects of their content. Aspects of their profile can be set to public or private, with public set details visible to any site user while private details are visible only to a select few on the site (Potentially collaborators or through a 'friends' system. The exact details of the implementation are still being hammered out). Media uploads will include an 'unlisted' option, where the media is not listed in search results or on the site front page but can be linked to and publically accessed.
User Harassment
Because users have the ability to comment on content on the site, there will invariably be a few who choose to abuse this system, to this end, the owner of a piece of content (media, profile, etc) can opt to 'block' a user. This will prevent the blocked user from interacting with the blocking user in any way, effectively making it as if the blocking user does not exist from the perspective of the blocked user. The blocking user may still be able to view content belonging to the blocked user, but will be unable to comment or interact with it.
TOS Violations or Rights Abuse

If a user suspects that another user has in some way breached the site TOS (such as uploading inappropriate content or claiming content they do not actually own), the user is able to issue a 'report' that will be investigated by the site admins for legitimacy. If the report is found to be legitimate, the administrators will decide on an appropriate response, whether that be taking down the offending content or banning the user outright. If a user repeatedly files fraudulent reports, they will be punished.

Risk Response

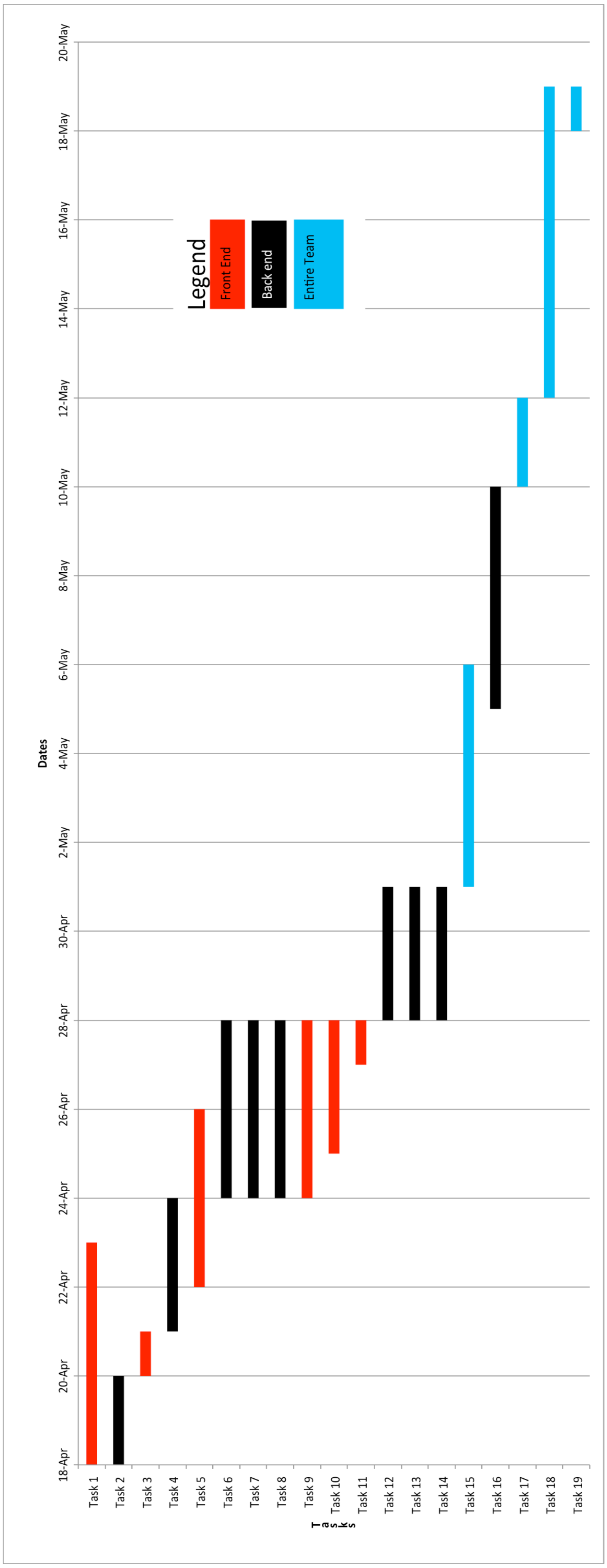
System Breach
In the unlikely (but always possible in today's world) even of a system breach that leaves any user data on the site exposed, all user passwords will be reset (requiring a password change on next login), any potentially exposed encryption keys will be discarded and updated, and all users will be notified that a breach has occurred and the steps taken to mitigate risks to their personal data. When such a breach occurs, methods will be investigated and implemented to close the security hole that allowed the unauthorized access to occur.

Database Damage
If somehow the database manages to become damaged (through malicious intent or otherwise) there are several courses of action we can take. In the case of smaller issues (such as corrupted transactions) a simple database rollback can be done to undo the damage, while in more severe cases, we can resort to restoring the system from a previous backup, while this would result in some data loss, under certain circumstances it may be unavoidable.

Project Timeline

Description	Start Date	End Date	Duration (Days)	Extended Description	Responsibility
Task 1	18/4/2015	22/4/2015	5	Complete UI design for the home page.	Front End team
Task 2	18/4/2015	19/4/2015	2	Allow user to sign up in Rails application.	Rohan
Task 3	20/4/2015	20/4/2015	1	Test the user sign up.	Front End team
Task 4	21/4/2015	23/4/2015	3	Allow user to sign in the Rails application (Sessions).	Rohan
Task 5	22/4/2015	25/4/2015	4	Complete UI design for the profile page	Front End team
Task 6	24/4/2015	27/4/2015	4	Allow music to display algorithmically and dynamically on the front page.	Rohan

Task 7	24/4/2015	27/4/2015	4	Allow users to maintain their profile, follow other users and view other profiles and show music on profile.	Vui Chee
Task 8	24/4/2015	27/4/2015	4	Create forms which will be used by the User to upload images/music with information.	Alex
Task 9	24/4/2015	27/4/2015	4	Work on implementing an instance of jplayer (audio player) which will play music on the homepage.	Josh
Task 10	25/4/2015	27/4/2015	3	Complete UI design when you click music/album + comments	Front End team
Task 11	27/4/2015	27/4/2015	1	Test all the functionalities implemented form Task 3 to Task 7.	Front End team
Task 12	28/4/2015	30/4/2015	3	Provide the functionality for users to rate music/download music +Donate to artists	Rohan
Task 13	28/4/2015	30/4/2015	3	Provide the functionality for users to comment on profiles/music and respond to personal messages.	Vui Chee
Task 14	28/4/2015	30/4/2015	3	Implement search functionality for users to search artists/music/profiles	Alex
Task 15	5/1/15	5/5/15	5	Allow users to maintain(CRUD) multiple playlist using cookies + sessions.	Back End + Front End teams
Task 16	5/5/15	5/10/15	5	Create a notification system to notify followers of new music + notify when you get a personal message.	Back End team
Task 17	5/10/15	5/12/15	2	Usability testing, White box testing, Unit testing, Functional Testing	Back End + Front End teams
Task 18	5/12/15	18/5/2015	7	Reserved for testing + Any changes that may need to be made as a result for testing	Back End + Front End teams
Task 19	18/5/2015	18/5/2015	1	Submit project!!!	Team



Project Timeline Chart

Due to the page limit, we can only afford to place the chart at the end of the report.