Developing ArtMatches: a Social Network for Artists

Kalin Stoev 2107114S

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements for the Degree of Master of Science at the University of Glasgow

Submitted: 22/09/2015

Abstract

Existing popular online social networks for artists do not facilitate collaboration and bringing together different artistic mediums. However, many works of art such as movies, comic books or other mixed art performances are collaborative in nature and need very different skill sets to produce. This dissertation deals with the process of designing, developing, testing and evaluating a new social network for artists, which addresses this issue.

Following a survey of existing solutions, the project was designed as a real-time app which meant that any change in its central database is instantly propagated to all of its subscribed clients. This was achieved using a relatively new web communication technology – WebSockets – which allowed uninterrupted communication between the client and the server and allowed the app to work as a single page application. The final design provided its users the ability to post, search, talk about, match, and edit art in a seamless fashion.

The product was evaluated iteratively by two groups of users to determine its usability and usefulness. The application was well accepted among potential users and scored above average on its usability tests. The results suggested that a production-ready solution would be of interest to artists.

The dissertation ends with addressing the application's shortcomings and gives suggestions for further work. Last but not least, it lists key takeaways from its development process from the developer's perspective.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Kalin Stoev

Signature: 

Acknowledgements

First, I would like to thank Leif Azzopardi, my supervisor, for his expertise, insight and advice throughout the proposal, development, evaluation and writing of the final project.

Second, I would also like to thank all volunteers that participated in the application evaluation for their time, invaluable feedback and comments that made the application better.

Finally, I would like to thank the Meteor community for all the help they have provided in the form of tutorials, books, and other learning materials. Without them, the development of this project would not have been possible.

Contents

Acronym	Meaning
AJAX	Asynchronous JavaScript and XML
BSON	Binary Structured Object Notation
CSP	Content Security Policy
CSS	Cascading Style Sheets
DDP	Distributed Data Protocol
DOM	Document Object Model
GIF	Graphics Interchange Format
HTML	Hypertext Mark-up Language
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
RPC	Remote Procedure Calls
SQL	Structured Query Language
SPA	Single Page Application
SUS	System Usability Scale
UI	User Interface
URL	Uniform Resource Locator
WAF	Web Application Framework
XSS	Cross Site Scripting

Table 0.1 List of acronyms used throughout the dissertation

Chapter 1 Introduction

"I found that it wasn't so oddball to like music and poetry and visual arts, they're kindred spirits." ***J. Carter Brown – Director of U.S. National Gallery of Art (1969-1992)***

1.1 Overview

This dissertation deals with a project whose aim was to create new software that would allow artists from different backgrounds to showcase their art and encourage them to collaborate and cross the lines between various artistic mediums. A web application was developed in several planned stages to meet the project's aims. The way this dissertation is structured is by following each stage of the software development lifecycle (Figure 1.1).

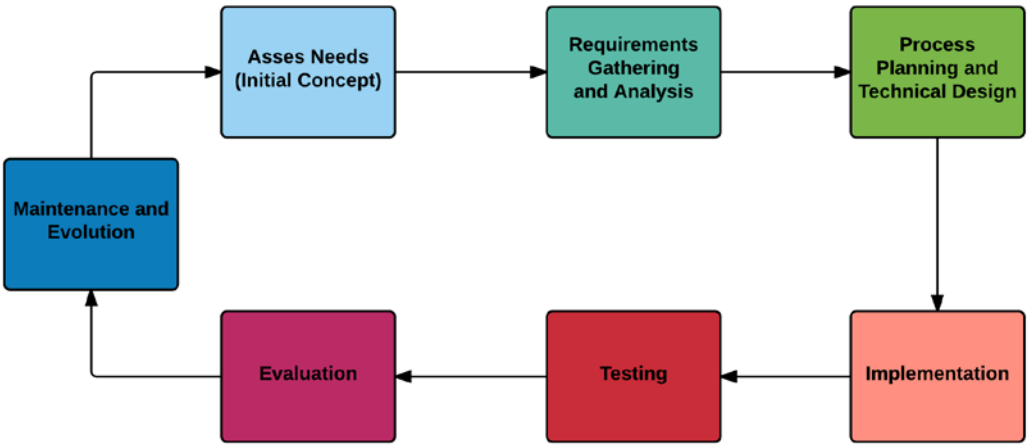

Figure 1.1 The stages of the Software Development Lifecycle

Each chapter covers what actions, techniques and outcomes were executed and produced to develop the application successfully. Furthermore, it lists any challenges, changes or shortcomings that had to be addressed and carried out throughout development to produce a robust, well-designed, functional, usable and maintainable product.

1.2 Project Context

The act of creating and experiencing art is an essential activity of every modern person's life whether it would be having a hobby as an artist, earn a living through art or just being a fan. The following dissertation would not deal in detail with the idea of what art is or any particular artistic practices in general. For the sake of discussion, when referring to art this proposal will refer to the sum of visual, written and musical arts. Furthermore, this particular piece of

work will talk about art as a social experience and will investigate and propose how it could be better translated into the web. For the rest of the dissertation, the current project will also be referred by its working name ArtMatches.

The best examples of social experience on the internet are the most popular social networking sites such as Facebook, Twitter, LinkedIn, Pinterest, etc. Each of these platforms differs in the way it allows its users to present themselves and interact with each other. However, the platform's core purpose stays the same to connect and engage users in a conversation: sharing personal photos or creative work, expressing opinions, talking to other members, etc.

Online social networks and the world of art are no strangers to each other. There is a plethora of online applications that create their own communities (deviantart.com, artstation.com, soundcloud.com), help teach art (ctrlpaint.com, artyfactory.com) or deal with the trading of art (etsy.com). The focus of this particular project will be on dealing with art primarily for pleasure and pursuing a creative passion opposed to a commercial gain. It also will approach the nature of art as a collaborative, collective experience opposed to a solitary, isolated activity.

Chapter 2 Problem Statement

Current online social networks for artists don't facilitate collaboration and matching of different artistic mediums. However, many works of art such as movies, comic books or other mixed art performances are collaborative in nature and need very different skill sets to produce. The problem which this project aimed to solve was to provide a web application that engages artists, encourages them to collaborate and to cross artistic boundaries.

An artistic collaboration would take hours of work and iteration that could require the presence of all participants and constant communication. However, the potential collaborators need to find each other first. Ideally, they would see if their artistic styles match and get a quick feedback to what extent others enjoy their work together. Thus, the notion of matching and collaboration emerge as necessary features for a social network targeted at artists. The best way to illustrate the discussed problem is to look in detail at existing solutions and where do they lack in promoting collaboration among artists.

Current online solutions follow the pattern shown in Figure 2.1.

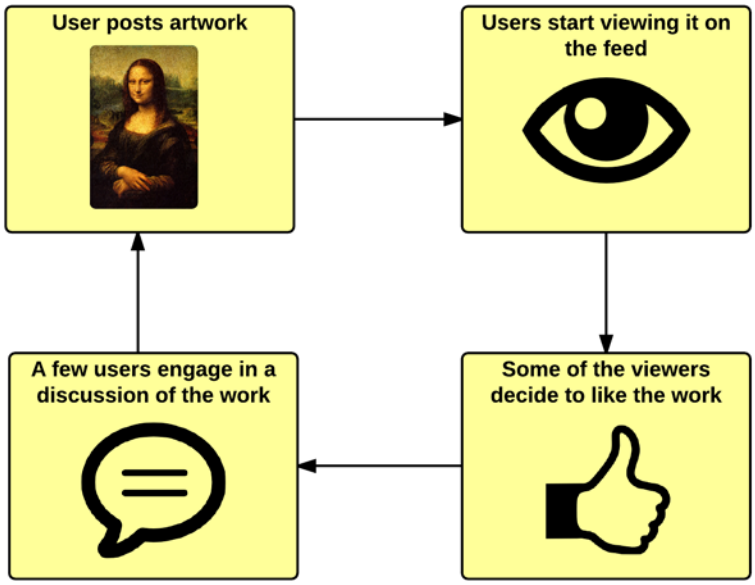

Figure 2.1 Typical user interaction on an art social network

Most comments are quite casual and just express praise about the quality of the work [1], for example: *"Don't think I need to tell u how brilliant this is but wow...BRILLIANT!"*. The problem of this pattern is that it doesn' facilitate collaboration or encourage users to reach to other artists. In most social networks, an artist who has a similar style has to either post a comment with a link to their profile or send a message (if available as a feature) to the artist and engage in a conversation separate from the work. What this project currently offers is a social network that takes what existing applications have and creates

a product where collaboration and matching of artists resides at the center of its design. Figure 2.2 shows the current interaction pattern in ArtMatches.

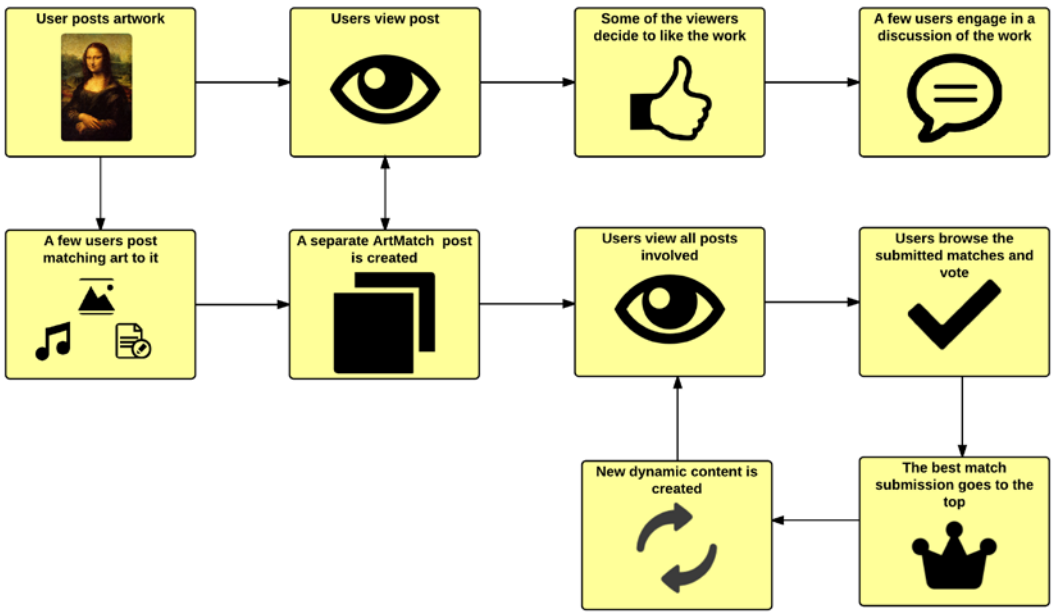

Figure 2.2 ArtMatches user interaction pattern

Examining the pattern in Figure 2.2, there is more engagement between the users. They are actively allowed to participate in the content creation and to cross artistic boundaries, instead of remaining passive or just expressing their opinion towards the artwork.

In conclusion, the current project solves the problem of art collaboration and matching artistic styles by providing a social network application that puts these at its core. It possesses the fundamental features of its competitors, but it also encourages artists from different disciplines to come together and engage in further collaborations.

Chapter 3 Survey

A survey was done of existing solutions to capture the requirements for the project. It analyzed three big social networks, evaluating their strengths and weaknesses. Next, it also explored an existing proof of concept that similar to the project's idea of matching art mediums.

The following survey looks at existing art social networks and learns what features are typical and useful and any lessons that could be learned from them. All of the following examples are well-established web applications with thousands of users. Consequently, the aim of this survey was not to uncover any obvious deficiencies but to gather what useful features they offered and how they could be incorporated in ArtMatches.

DeviantArt is the biggest and most feature rich of the analyzed networks. To avoid repetition, when analyzing ArtStation and SoundCloud only unique features are highlighted. At the end of this chapter, a comparison table offers an overview of what features were chosen to be included in the project.

3.1 DeviantArt (deviantart.com)

In the online realm of art, there is one web application that stands out, in size and popularity, over anything else - DeviantArt. As of 2015 it is the largest art community on Earth with 26 million registered members and 251 million artist generated works of art [2]. Academia has studied the application's community and evolution [3]. Additionally, the entertainment industry has used it as a promotion platform to create contests for popular products such as Tomb Raider, Riddick, Dark Shadows [4].

DeviantArt offers a platform for its users to share and discuss a wide variety of art from drawings and photography to artisan crafts and literature.

3.1.1 Strengths

The main advantage of DeviantArt is that it has successfully achieved to be a digital ecosystem where a variety of different art forms coexist together effectively organized into categories and groups. The sheer volume of the community also offers a great opportunity for everybody to share their art and receive feedback from thousands of fans.

A thorough examination of DeviantArt revealed these particular features that define the functionality of the application:

1. Each user has a profile that has the following features:
    - short description of the user;
    - a personal gallery;
    - a collection of favorites;
    - option to sell their art as print;
    - option to follow other users and get notifications when they post anything new;

- option to post status updates similar to Facebook and the ability to have their personal blog

2. Users can create interest groups and new discussion threads in a dedicated online forum
3. Users have chat rooms and the ability to send each other private messages
4. User can create and organize themselves in art groups
5. A post allows a single piece of artwork (image or text), has one author and allows other users to comment or to offer a critique through a specialized interface.
6. All art is organized with hashtags and categories
7. Users receive notifications for events generated by other users (such as a comment, a following, etc.)

3.1.2 Weaknesses

DeviantArt is excellent at what is supposed to be: a thriving online art community. Where it falls short is that it approaches art creation as a solitary activity. A post (called a deviation) could have only one author, one piece of work and it doesn't allow other users to interact more than just commenting or critiquing.

It only allows seeing page views on the user's profile but not on any individual post. There is no option for the users to like a post.

Another problem of the website is that has too many features. Users who are not proficient in English, and thus unable to fully comprehend the tutorials, have trouble using the website and get lost in the interface. A simple more niche application could be able to solve that.

Finally, DeviantArt doesn't allow posting of any audio files which has been a continuous complaint from users.

3.2 ArtStation (artstation.com)

ArtStation is a typical example of an app that is niche and focuses only on visual arts with a big accent on professional networking and jobs in the entertainment industry. Although the software is still in its beta version, it has been getting a lot of attention from visual artist professionals, especially in the gaming industry.

3.2.1 Strengths

ArtStation has a simple and very intuitive interface its post feed encompasses most of the page encouraging easy exploration of user created content. Each post allows for multiple images or embedding of videos. Other users could like the post and see how many page views the post already has.

The website has a dedicated jobs section, and each artist could also create a personal portfolio with a professional look in a matter of one click. Users are also

able to disclose their current occupation/job title, so it makes it easier for others to spot potential connections. Each user could denote on their profile if they are open for hire.

3.2.2 Weaknesses

ArtStation is highly specialized for professional visual artists in the entertainment industry. For many amateur artists, especially those interested in music or writing, it would be almost impossible for them to join and interact with the rest of the community. The application doesn't support private messages and for contact it just offers the Twitter or Facebook profile of the user. It is very niche, and it acts more as a matching app between professionals and employers than between individual artists looking to collaborate with each other.

3.3 SoundCloud (soundcloud.com)

SoundCloud is one of the largest audio distribution platforms in the world with 40 million registered users [5]. It also acts as one of the most socially active places where musicians could interact with their audience, meet others and talk about their art. The network has extended to offer content such as notable speeches (including Barack Obama's), audio shows, interviews, etc. Sometimes the website is referred as the YouTube for audio because of its sheer volume.

3.3.1 Strengths

SoundCloud is praised for its ability to share audio easily with a simple drag & drop interface in the browser. It allows reposting the work of other artists in the personal user feed. The app allows timed comments on each track that appear on a sound wave segment, chosen by the user. It is considered very useful when artists share their tracks privately for collaboration and feedback.

3.3.2 Weaknesses

SoundCloud doesn't offer much for users to talk about themselves online besides a very short description. User interaction is limited to comments, likes, following, messaging and reposting. Again, if one wants to collaborate with another artist, there is no dedicated feature for it. Furthermore, the website is highly specialized, and it caters only to musicians.

3.4 Brain Pickings (brainpickings.org)

Maria Popova and her blog brainpickings.org which she describes as "a library of cross-disciplinary interestingness and combinatorial creativity" is the last example on the web that this survey examined. Unlike the other three websites this one is not a social network, in fact, it doesn't even allow user discussions and comments. What makes it relevant and unusual is that it illustrates perfectly how different artistic mediums could work together and produce engaging content just by being grouped and presented together.

Maria Popova runs a little side project called Literary Jukebox (Figure 3.1) which pairs passages from books and quotes from famous people with music [6]. Additionally, she tends to include vintage illustrations or famous paintings that

fit the topics of her posts. Her blog is extremely popular, and her ability to pair art is a big part of it. Thus, this was used as a proof of concept for ArtMatches. It suggested that the application could help artists not only to match and collaborate but also to create new content just by the act of meeting each other and presenting the process to others.

Figure 3.1 Literary Jukebox by Maria Popova

3.5 Summary

The following table (3.1) summarizes how the main features of each social network differ from each other and what features were chosen to be implemented in the project.

Features	DeviantArt	ArtStation	SoundCloud	ArtMatches
Personal Gallery	✓	✓	✓	✓
Create and see posts	✓	✓	✓	✓
Comment on posts	✓	✓	✓	✓
Event notifications	✓	✓	✓	✓
Like posts		✓	✓	✓
User following	✓	✓	✓	✓
Generate personal portfolio		✓		
Create art groups	✓		✓	
Allows multiple art mediums	✓			✓
Sell art	✓			
Hashtags to organize art	✓	✓	✓	✓
Art categories	✓	✓	✓	✓
Search content by keywords	✓	✓	✓	✓
Drag and drop file upload	✓	✓	✓	✓
Group Chat	✓			✓
Private Messages	✓			✓
Multiple artworks per post		✓		✓
Submit a matching artwork to another user				✓
Multiple authorships per post				✓
Real-Time				✓

Table 3.1 Features comparison between existing Apps and ArtMatches

It is worth noting that none of the analyzed applications provided a full real-time user experience. Currently the most popular online social network in the world is Facebook, approaching 1 billion users [7]. In fact, in countries with heavy internet usage like the United Kingdom, more than one-third of the whole

country population visits the application [8]. Facebook offers a real-time experience and sets certain expectations to other social networks. Therefore, any new or existing social network could be driven towards being real-time and meeting the demands of users who are used to live interaction compared to the old model of static web pages and frequent reloads.

The proposed project decided to provide a real-time user experience right from the application's inception because it fit well with ArtMatches's overall purpose to encourage artists to interact and collaborate with each other. Real-time could allow in future releases for users to draw together on the same canvas, collaborate on the same written work similar to Google docs, brainstorm ideas online or even jam with music from remote locations.

Chapter 4 Requirements

All requirements for ArtMatches were defined in natural language similar to what Sommerville refers as "user requirements", which are high-level abstract requirements that define what the system should be able to do [9]. Furthermore, defining the user requirements helped in developing testing cases later during the acceptance testing and the evaluation of the application. The requirements were divided into functional (what the system should do) and non-functional (what the system should be).

4.1 User Scenarios

Prior to writing any detailed requirements, two user scenarios were sketched out to allow for better understanding of who, how and why is going to use the application. These proved to be useful in gathering the initial functionality and separating users in two main profiles – artists and fans.

Scenario 1 (the artist): Jenn is a 22-year-old, fine arts graduate from London. Although she knows quite a lot about classic art, recently she has started experimenting with digital graphics of people and landscapes. Most of her friends are not much into that particular scene, so she needs a place to meet other like-minded people and to showcase her art. She heas about ArtMatches through a friend and after showcasing her art for a month, she receives some matches from other graphic artists matching her style, and they start talking on the platform. She meets one of them offline, and they start working on a small project together involving a series of interconnected pieces of art. Meanwhile, some writers liked one of her paintings and had submitted a short story based on it. She loved it and had started thinking of turning her small collaboration into a bigger team and maybe coming up with a graphic novel that could boost the visibility of all artists involved.

Scenario 2 (the fan): Bob is a 28-year-old and works as a software developer in a big corporation. He has always been quite keen on music but never had the time or the patience to learn a musical instrument. Recently he has been browsing on ArtMatches after work for new songs submitted by the community and has been very curious to see what songs other musicians send to each other as matches. He has been voting on each match he likes, especially when it comes to bluegrass music which is his favorite. Bob is very happy that although he is not making music himself he is helping other people to go up in the ranking. Moreover, he hopes to hear some of them do a jam together with his favorite artists on the network. As an extra fun, he has been going through some of the written matches and has been thinking if any of them would make some good lyrics to any of the songs.

4.2 Functional Requirements

The initial requirements for what functionality ArtMatches needed to have were gathered based on the survey of existing solutions in Chapter 2. Next they were prioritized and detailed in a MoSCoW analysis (please see Appendix A). This particular type of analysis allows an easy way to prioritize the features that are critical to the core functionality of the application. All work was structured in such a way to deliver those features first ("must haves" and "should haves").

One of the potential issues outlined in the initial proposal for the project was that ArtMatches would require a significant amount of work and knowledge to implement all of its proposed features. The whole application was developed by me and I had to attain an advanced working knowledge of HTML 5, CSS, and JavaScript programming both for the client and the server. As a result, one of the big risks was falling behind schedule and not delivering all key features.

All "must haves" were implemented in the application and some were changed from the initial proposal to improve the user experience. More than half of the "should haves" were successfully implemented and all of the "could" and "would haves" were left out of the final product due to time constraints. The implementation section will discuss in detail how the core functionality was delivered in the development of ArtMatches.

4.3 Non-Functional Requirements

The non-functional requirements for ArtMatches were related mostly to enhancing the user experience and allowing for a broad range of users and device types. A detailed list of the requirements is shown in Table 4.1.

Summary	Detailed Non-functional Requirement
Cross-device compatibility	The application should be able to run and be compatible with various screen sizes from really small (smartphones) to tablets and big monitors of desktop PCs.
Data security and user privacy	The application should be secure and keep its private user data in an appropriate manner for a publicly accessed application
Seamless UI experience	The application should feel quick and seamless to the user and handle page loading in an acceptable fashion (under 1000ms).

Real-time UI experience	The application should be real-time which means any input from the user should be reflected on the UI instantaneously (under 100ms for reaction).
Usability – no previous training required to use the app	The application should be easy to use, and users shouldn't need extra help to be able to interact with it in an effortless way right from the start.
Incremental approach towards the user interface design	The UI of the application should work with queues for its interface mimicked from other social networks that provide similar functionality (see Chapter 2)
Dissertation time-constraint	The application should have all of its core functionality developed and tested in 60 days

Table 4.1 ArtMatches non-functional requirements

4.4 Use Cases

Various types of functionality were developed in order to make ArtMatches a working final product. To keep track and potentially be able to communicate all the features to others a use case diagram was developed which covered all the functionality of the application.

The diagram followed the standard UML notation. It also makes use of a hierarchy of functionality such as extend and include defining explicitly the dependencies between the different types of functionality (Figure 4.1). The use case diagram was used throughout development and later user acceptance testing to ensure that all planned functionality was delivered and tested.

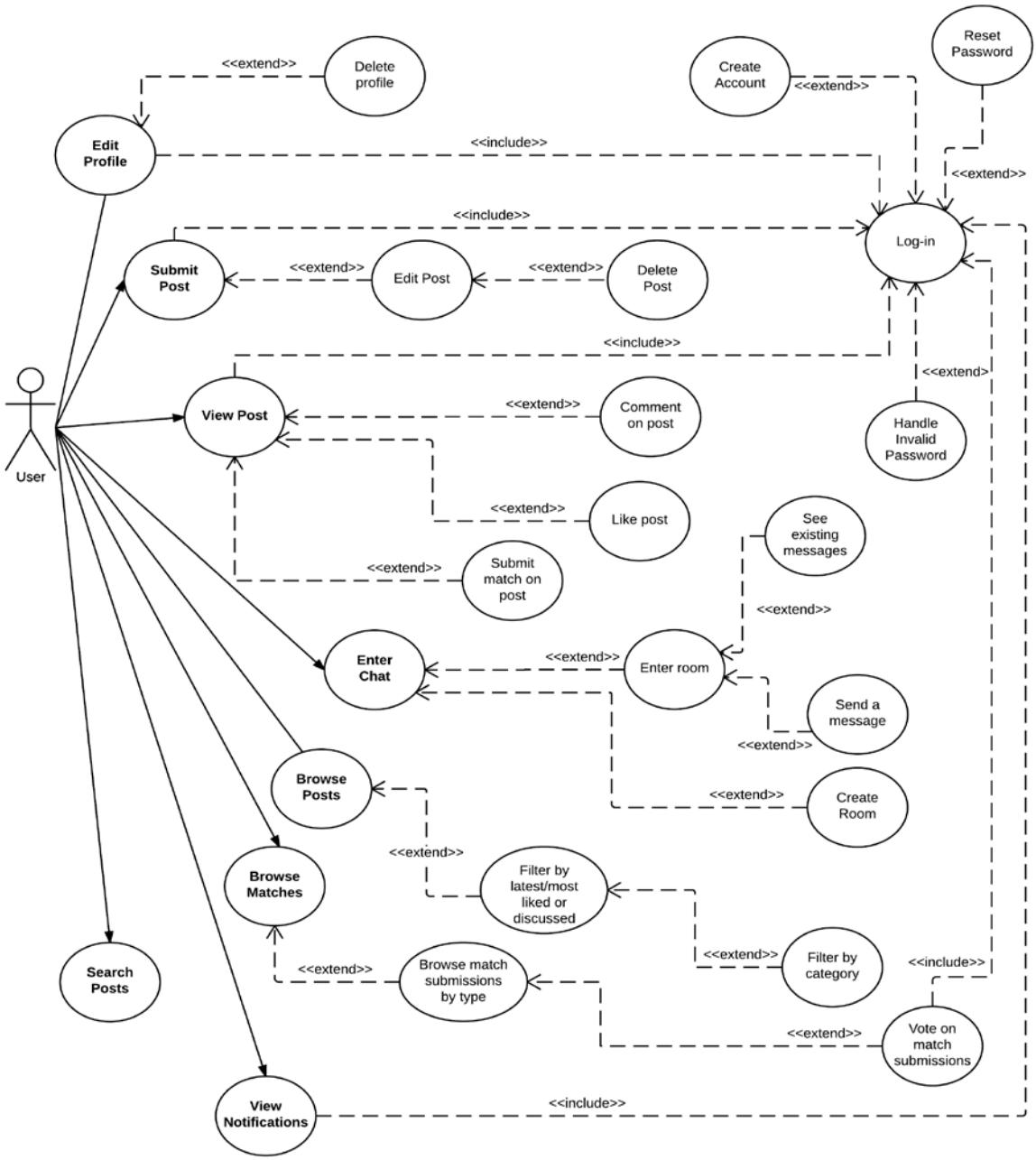

Figure 4.1 ArtMatches use case diagram

Chapter 5 Design

The first step in ArtMatches' design was to choose what development route the application would take. All the previously surveyed solutions, such as DeviantArt, were created as web applications. This fact gave a significant weight to going this way because it would allow for different devices and operating systems while using the same technology (developing for the browser). However, two of the non-functional requirements were that the application should be real-time and seamless which could be difficult to achieve with the traditional request/response model for the web. On the other hand, if the application was developed as a native to each platform (Windows, Mac OS, Android, iOS - e.g. Spotify) then it would take way longer than the sixty days allocated to development. After careful consideration, the short time constraint and the ability to deliver the service to all platforms at once resulted in going the web application route.

5.1 System Architecture

ArtMatches was designed as a 3-tier web system with a thick client (Figure 5.1). Each tier offered a clear separation of concerns. The client handled what data the user sees and how it was presented. The middleware was responsible for controlling the access to the data and carrying out any modifications while the database handled storing and querying the data. The separation in tiers allowed easy scalability in the future by being able to replicate them across multiple machines without replicating the whole application.

One of the main changes from the initial design was adding a second external database in the form of Amazon Simple Storage Service (S3). The application deals with storing and serving back to the client a significant amount of data because of all the image and audio content that is inevitable when dealing with art. Consequently, this required a service that was well specialized in serving data to clients from around the world and also allows a lot of free storage since the project had no budget. The ArtMatches server was to be located in one place around the globe due to the small scale of the app and would not be able to accommodate for serving content equally fast to all clients geographically. As a result, S3 was chosen which also offered integration with CloudFront a content delivery network that deals exactly with serving content to the client from the closest server possible.

ArtMatches was integrated with four other social networks. It was designed, so users were to be allowed to embed audio from other social networks such as YouTube and SoundCloud which helped to reduce the load of data going between the client and the rest of the tiers. To facilitate user registration ArtMatches was also integrated with Facebook and Google+ so users could create a new account with just one click.

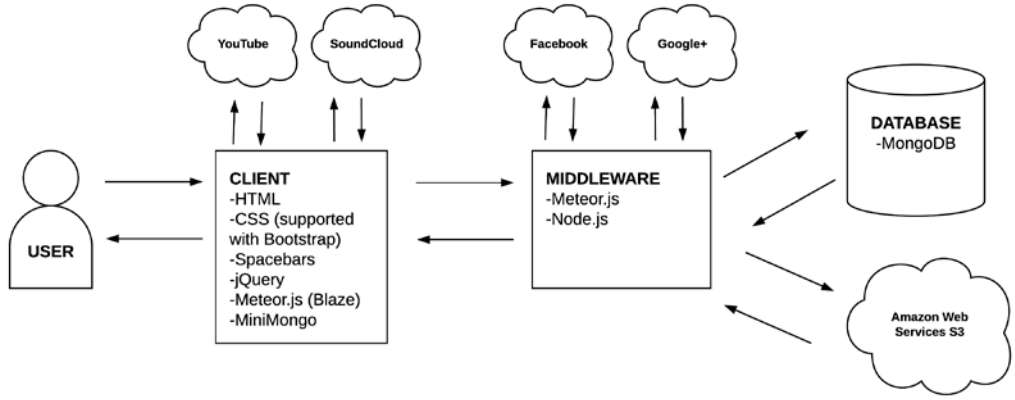

Figure 5.1 ArtMatches system architecture

5.2 Entity-Relationship Structure

An Entity-Relationship Diagram (5.2) was created early in the design process to facilitate the implementation of the underlying data model for ArtMatches and guide development. All entities remained the same except for a new entity added later in development called "Submission." It was added as a "bridge entity" between the "Post" and "ArtMatch" entities to tackle the many-to-many relationship between the two which was difficult to manage otherwise.

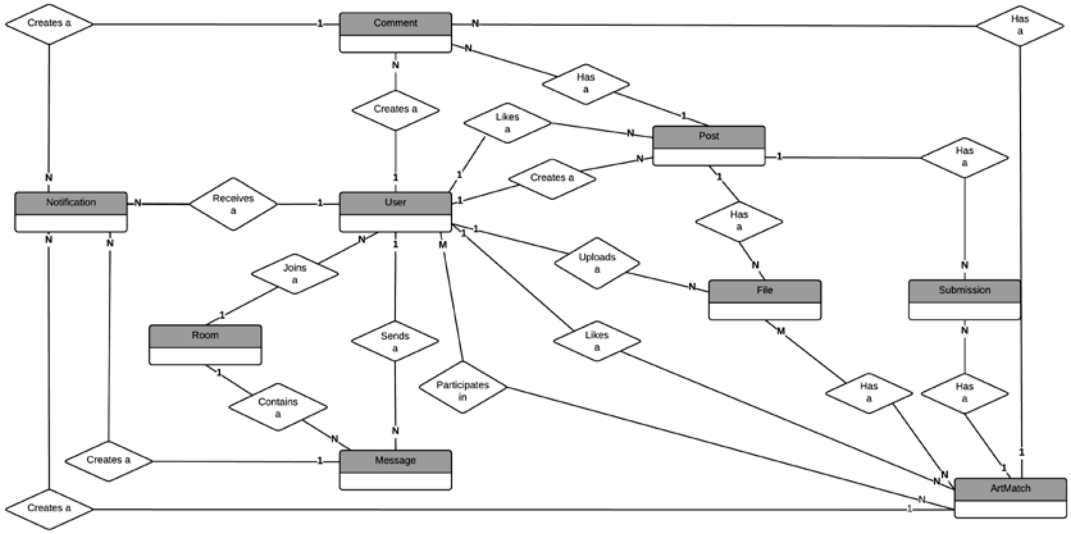

Figure 5.2 ArtMatches entity relationship diagram

5.3 User Interface Design

During the interface design stage, all decisions were made in line with the application's requirements. Its interface had to be self-explanatory, with no user manuals, aesthetically pleasing and adaptable to different devices. Furthermore, it had to be designed in such a way that it would allow the user to accomplish each of the use cases outlined at the beginning.

5.3.1 Graphic Design

The overall look and feel of ArtMatches evolved during development reaching its final black and white minimalistic design. The color palette and the square sharp-edged design of the different components were derived from the Twitter Bootstrap Yeti theme, available from bootswatch.com [10]. Furthermore, some of the colors were changed. For example the background was modified from white to light gray which created a better feel for the application and put less strain on the user's eyes. A lot of icons were used throughout the design of the application. They allow users to perceive quickly the functionality and context of each UI component instead of relying just on text labels (Figure 5.3)

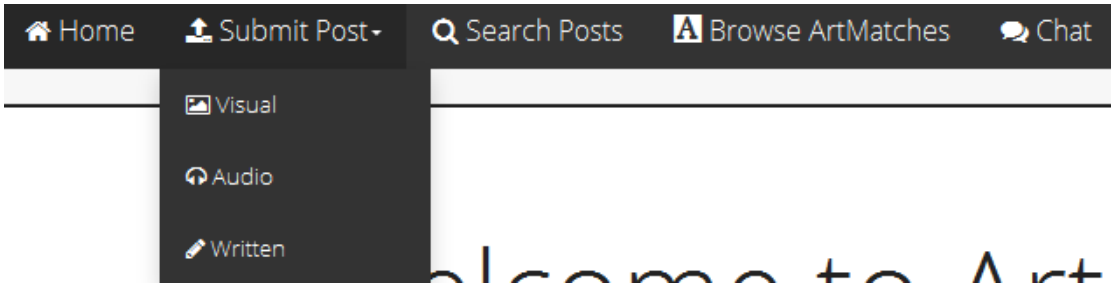

Figure 5.3 ArtMatches navigation icons

All icons were taken from the free, popular iconic font FontAwesome [11].

The final design of ArtMatches looks relatively minimal compared to other apps like DeviantArt (e.g. everything except the buttons is in the black-grey scale, not many popups or extra panels with recommendations, advertisements, etc.). There are two main factors that led to this design.

First, the application is based on user-generated content, and it seemed right to put the accent on the submitted art and not the application itself – the interface should be intuitive but not too flashy.

The second reason was more practical. There was very limited time to develop both all the functionality and the design of ArtMatches by a single developer. Thus, it would have taken away of the functionality and the testing if any more effort was spent on making the interface more elaborate. As a result, the design was chosen to be less elaborate or colorful. However, it still aimed to be usable

and aesthetically pleasing which was to be confirmed later in the user evaluation.

5.3.2 Layout

The user interface design of ArtMatches was heavily influenced by the standards used by the underlying CSS framework – Bootstrap. The whole layout was designed using a 12 grid system where elements are arranged in rows, and each could take between 1 and 12 columns. This is a useful way of structuring the interface as it allows a design that could be customized for different screen sizes. For example, each post item would occupy 3 columns in large screens and continually expand to 12 columns (the whole row) for very small screens (mobiles). The home page of ArtMatches was designed using two navigation bars. One was the main header that was present anywhere in the application and acted as the main navigation.

A decision that took some work was whether to include a home link since the logo was clickable and pointed to the home page. A survey of existing traditional networks resulted in adding a home link as well (Figure 5.4). Additionally descriptive labels were added to each icon and link to make it even easier for the user to navigate quickly around the application.

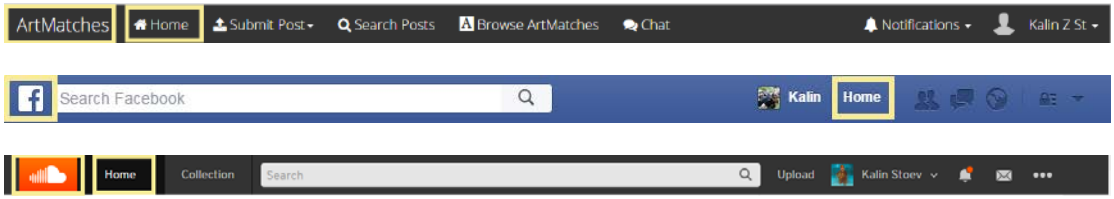

Figure 5.4 ArtMatches navigation compared to Facebook and SoundCloud

The second navigation bar was only available on the homepage was added to allow users to filter posts according to their interest (Figure 5.5).

Figure 5.5 Second navigation bar for filters

The application was designed to give constant feedback to the user if there are any errors based on the user input or if the action was successful. ArtMatches uses an open source JavaScript library – Toastr [12] – to style and render all the notifications (Figure 5.6)

Figure 5.6 ArtMatches notifications – success and error

5.4 Interface Screens

One of the primary goals for ArtMatches was to be real-time and to allow a seamless user interface. The current trend for creating a faster, seamless experience on the web is by using the single page application architecture [13].

In a standard web application any time the client state changes – for example the user clicks a link – the client sends a request to the server that returns an HTML page with data from the underlying data model. On the other hand, in a SPA all the client-side code (HTML, CSS, and JavaScript) is loaded at the initial connection between the client and the server in a single page. Thereafter, the DOM is updated continuously without the need to re-render the page. In order to have different screens (states), templates are utilized, and the constant asynchronous communication between the client and the server is handled either by AJAX requests or by using WebSockets. The transition between screen is handled by an application router which loads the appropriate template based on the current URL.

ArtMatches is organized in 15 screens that the user could see and interact with (Figure 5.7). Each screen contains several UI components that allow the user to navigate, read data or to interact with the application. Furthermore, every screen was created with the use cases in mind. Therefore, the sitemap of ArtMatches overlaps with its use case diagram. This allowed for better code structure and to avoid content coupling during development.

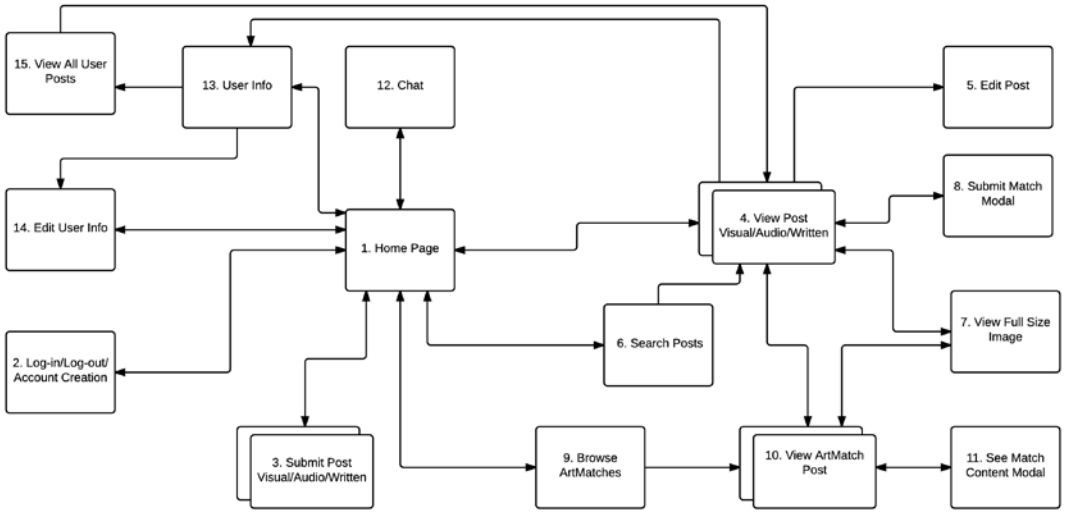

Figure 5.7 ArtMatches sitemap

The interface design and the information structure of ArtMatches took a top-down approach where it expects the user to land first on the homepage and navigate from there instead of expecting the user to land anywhere in the app. As a result, the main content was focused on the home page and so is the main browsing functionality. However, because ArtMatches is a SPA, it was only prudent to include the main navigation bar on each page. This provided an easy way for the user to either go back to the homepage or access the rest of the app functionality. Some of the major screens were wireframed early during development while others were implemented without any prior sketching.

5.4.1 Home Page

The home page screen of ArtMatches acts as the main screen of the application allowing the user to browse existing content. The second navigation bar allows filtering the content according to each individual's interests. The home page was wireframed early in development (Figure 5.8A) and changed several times based on user feedback and some small concept changes until it reached its final design (Figure 5.8B). One of the extra additions was the welcome banner that was introduced to act as a visual separator between the main navigation bar and the second navigation bar which is used to filter posts. It also serves as an easy visual cue to the user that they are on the home page.

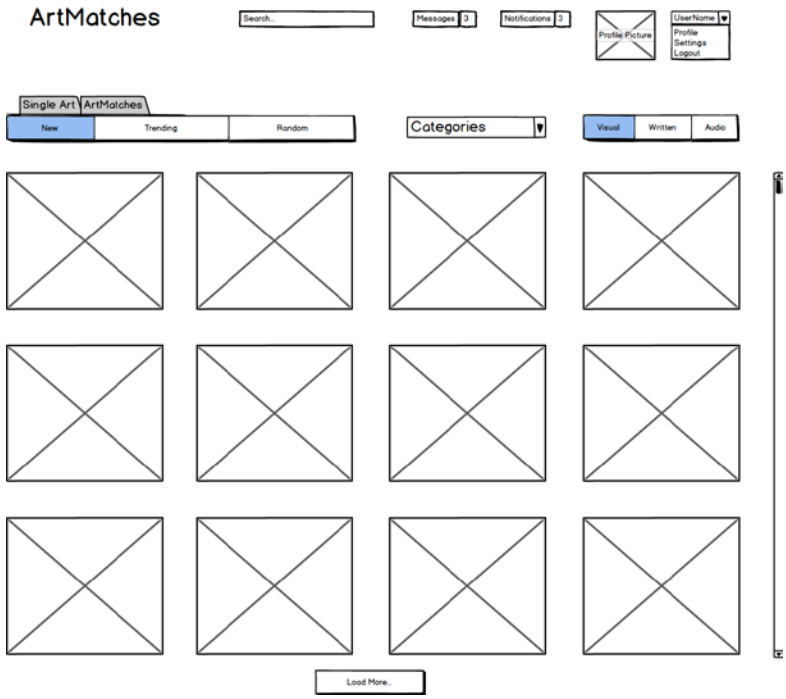

Figure 5.8A. Home Page – wireframe

When developing the second navigation bar, one of the significant changes from the wireframe was switching from tabs for art type to a select menu that was deemed easier and more intuitive during implementation. Also browsing ArtMatches and the searching function were separated in their separate screens mostly because it was getting way too complicated during the implementation to manage all templates on the same screen.

The last component on the home page is the "post item". It is a post template which tries to resemble a black picture frame with the title of the post and small icon indicating its type on the top, the author's name as a link at the bottom left and its number of comments and likes at the bottom right.

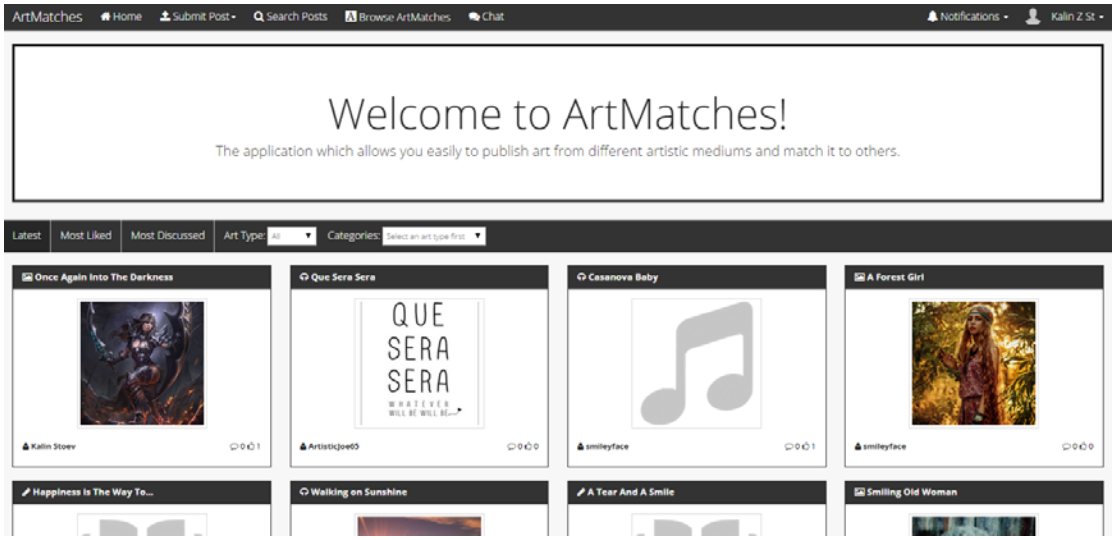

Figure 5.8B. Home Page - final design

5.4.2 Log-in/Log-out/Account Creation

This particular view doesn't take the whole screen but acts as a dropdown menu from the main navigation bar and has three different states – logged out, logged in and account creation. During the design, it seemed best to allow the user to access that key functionality from anywhere and thus it was all incorporated in the header instead of separating it into an another screen.

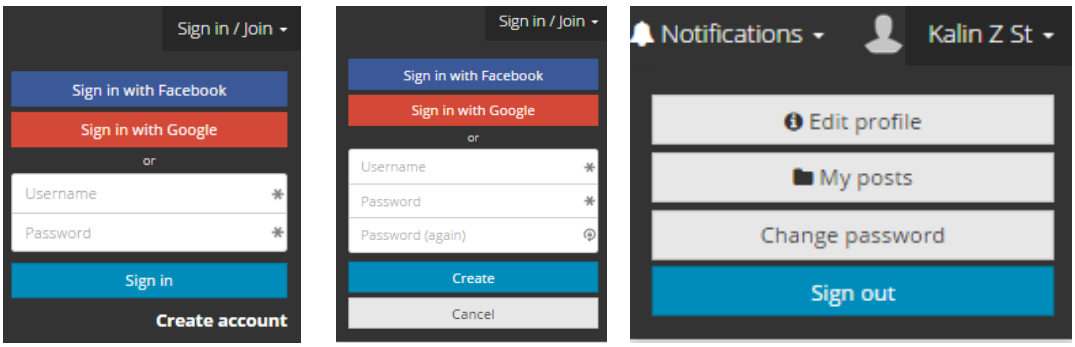

Figure 5.9 Logged out/Account creation/Logged in

5.4.3 Submit Post (Visual/Audio/Written)

This screen has three variations for each art type – visual/audio/written. Each post has to have a title, a category, and content. Furthermore, the post could have a description and a category. The screen for images and audio are almost identical. The main difference is that one user could upload multiple images per post for the visual whereas the audio screen allows only one audio file or embedded audio from another social network per post. The submit written post provide extra functionality with a text editing interface. It enables the user to

format the text in paragraphs and new lines with the option for additional styling (font size/type, font weight, etc.)

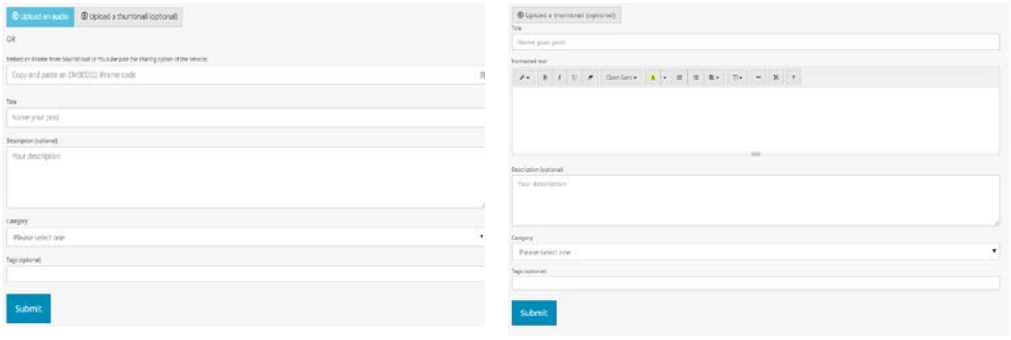

Figure 5.10 Submit audio and submit written screens

5.4.4 View Post (Visual/Audio/Written)

The view post screen again is slightly different for each art type. The visual would display all the uploaded images attached to the post. The audio would show an audio-playing file or embedded audio from YouTube/SoundCloud. Finally, the written one would display a piece of formatted text. The rest of the screen is the same for all. It shows the description of the post (if available) its details (such as author and tags). Next is the Submit Match button that was made extra big to draw the user attention. The like button was initially blue, but it looked very similar to the one of Facebook which confused some users. As a result, its color was changed to green in the final design. Underneath is the comment functionality that has a text area where the user could type their comment and a button to add it.

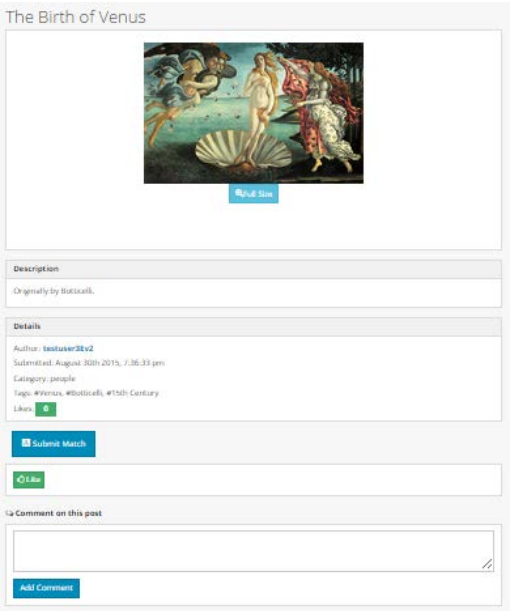

Figure 5.11 View post screen

5.4.5 Edit Post

The edit post looks almost identical to the submit post screen. The main difference is that when the screen is loaded all the input elements are preloaded with the data of the post being edited. At the bottom of the screen, the user has two buttons one to save the changes and another giving her the option to delete a post.

5.4.6 Search Posts

The search posts screen has just one input box which upon entered input automatically comes up with any matching results. This particular screen wasn't difficult to implement but presented some challenges in integrating with the rest of the app. Initially, it was designed to be part of the homepage but this presented a problem with rerouting upon changing the page state based on the filters in the second navigation bar. Currently, it is separated in its own screen which might not be the best solution but it does provide the functionality as expected by the user.

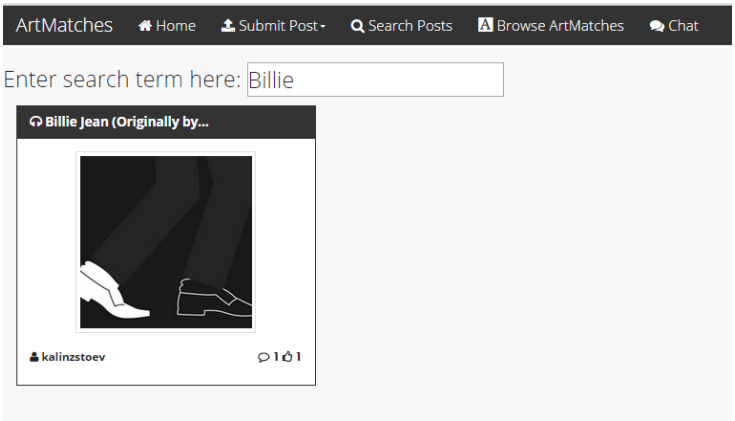

Figure 5.12 Search posts screen

5.4.7 View Full-Size Image

This is a very simple screen which just enlarges the image to its full size to take the whole screen and renders two buttons on the top of the image and on the bottom which act as links to go back to the post. The reason this screen was designed was to allow a seamless experience for the user. Opening the image in a separate/new tab and going back caused the whole application to reload which was unnecessary and cumbersome.

5.4.8 Submit Match Modal

The Submit Match deals with a core functionality of the app, and it was imperative that its design reflected its high priority. Initially, the natural way would have been to go with a new template on a separate route similar to how the search posts screen was designed. However, the interface design had to feel seamless and, therefore, that felt like a patchy solution. As a result, a more elaborate solution was implemented using a modal window instead of a separate template.

Modal windows are usually used to draw attention to the user about certain information on the screen or to confirm an action (e.g. a deletion, submission) while preventing the user from using the main screen. In the case of ArtMatches, the submit match modal had more functionality implemented which allowed the user to pick any of their existing posts content and submit it as a match. This approach provided a better user experience, especially for a web app, it felt very close to what a native application would look and feel like. However this provided some challenges in terms of implementation and presentation. To present the

information in an organized manner, three tabs were used to separate each art type. One of the deficiencies of the current design is that it uses a simple select radio button for each post's content. It would have been better if it highlighted the whole element on click. However, this particular solution required a lot of work to implement while not significantly altering the user experience and, therefore, was discounted and noted to be implemented in the future.

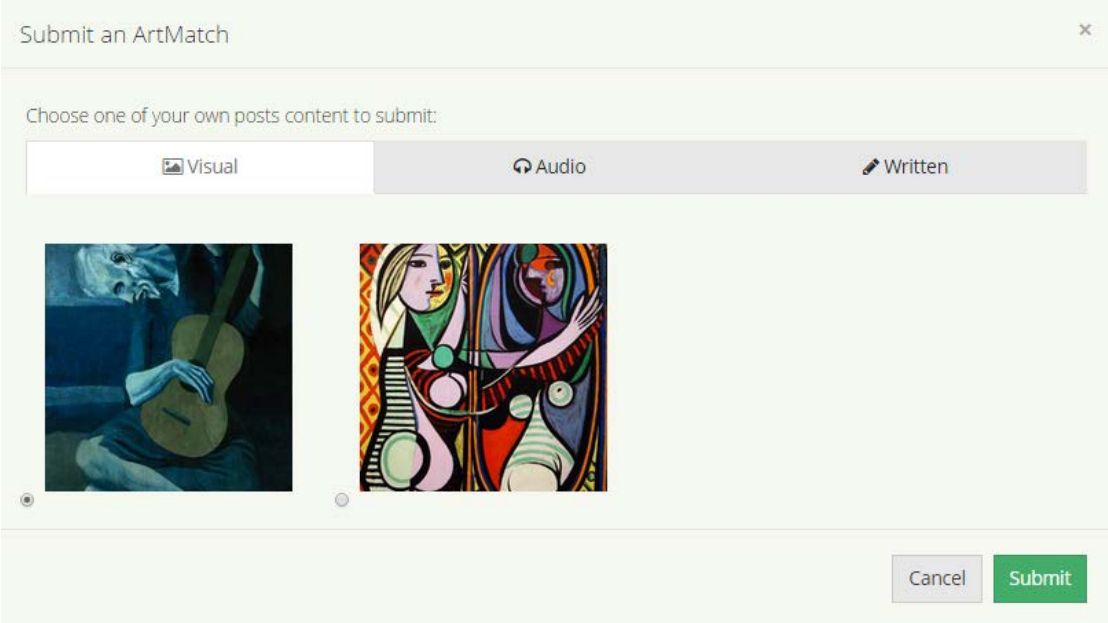

Figure 5.13 Submit match modal

5.4.9 Browse ArtMatches

This screen allows the user to browse through all existing ArtMatches. It behaves quite similar to the home page and the view post screen using the same template for any individual post item and a click on it would load the View ArtMatch Post screen.

5.4.10 View ArtMatch Post

The main novel feature in ArtMatches compared to existing solutions was the idea of having diverse user-submitted art and the ability to display it on the same page to create a richer content. Furthermore, the crossing of artistic boundaries encouraged collaboration. Figure 5.14 shows how the concept evolved from its initial conception until its final implementation. A significant change from the initial design was adding a match for the same artistic medium. This came about because it made more sense to a user to also have matches from the same art type (e.g. if you are a painter it would be natural to want to match art with other painters).

The initial design for this functionality allowed the user to browse through all submissions and be able to approve one submission from each art type to be displayed. However, during the later stages of the design, a potential user expressed interest in being able to see all the submissions which didn't get approved. As a result, the functionality was altered so the approving mechanism for matches was removed and instead a new voting system was implemented for each match (similar to Reddit). It was decided that a down-voting function won't

be implemented because artistic people could get very emotional about criticism towards their art and also it wasn't adding any significant value to the user experience itself. The new design added the benefit that the author didn't have to approve each match that could have been a lot of work and made the application even more interactive.

Figure 5.14A Initial concept design of an ArtMatch page

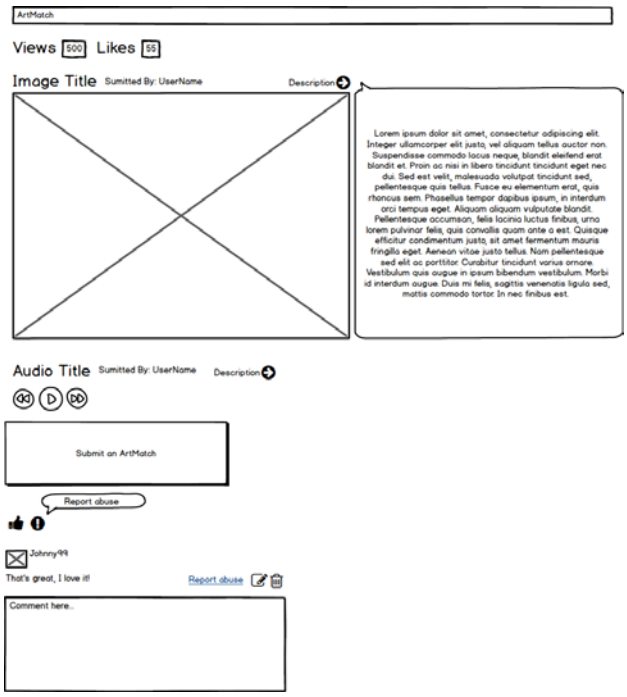

Figure 5.14B ArtMatch page - wireframe

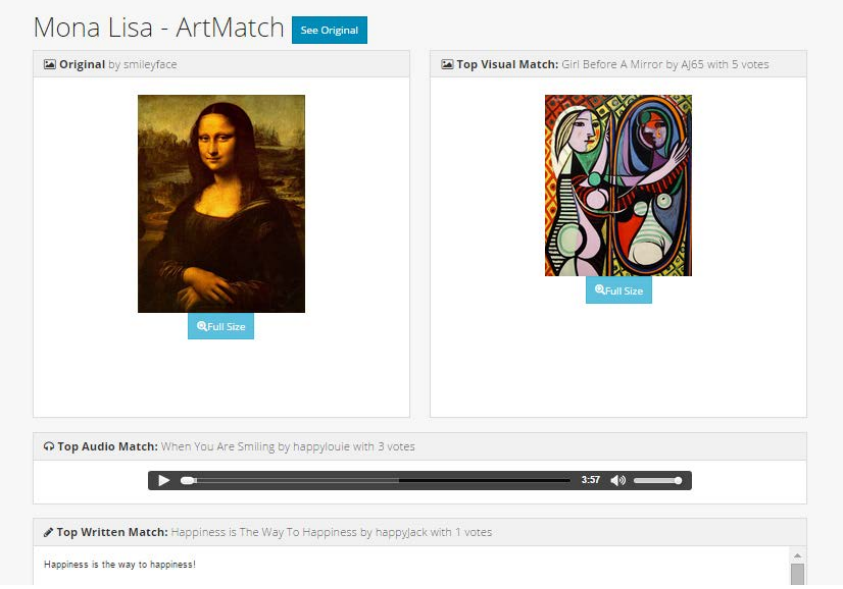

Figure 5.14C ArtMatch page - final design

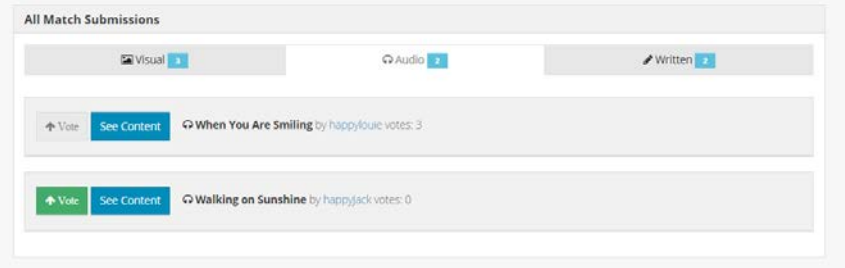

Figure 5.14D ArtMatch page voting – final design

5.4.11 See Match Content Modal

The see content screen is very similar to the view full-size screen in its functionality. One of the challenges in developing ArtMatches was how do you display rich visual, audio and written content without overcrowding the screen of the user. On one hand seeing each match right way could make it easier to browse quickly through all available matches. However, if the number of matches grows, the user screen would be crammed making her scrolling up-down and leaving the interface feel clunky. Consequently, all content was moved to a separate modal and that had its pros and cons. It did allow the user to eliminate all other distractions and focus on the art itself, but it did force him to make that extra action (click on a button) to browse the content of all matches.

5.4.12 Chat

The chat screen handles all the group interaction in the application. It allows the user to create new group chat rooms and to exchange messages with other users in real-time.

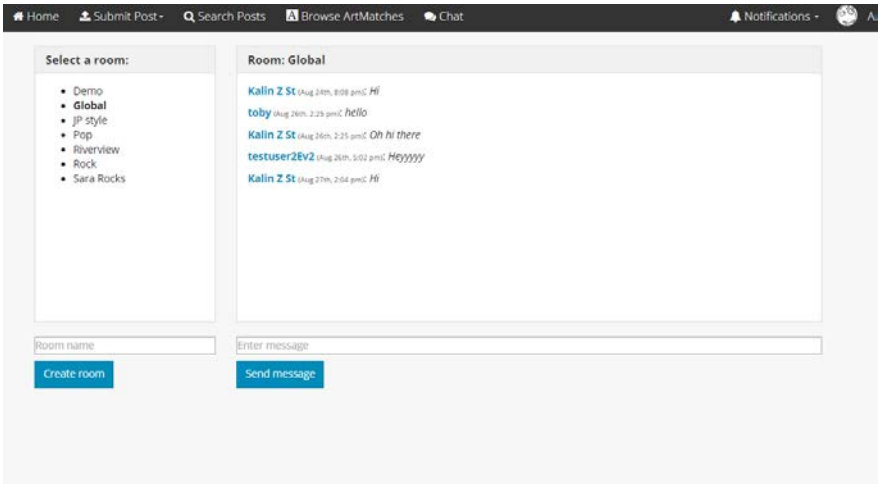

Figure 5.15 Chat screen

5.4.13 User Info

The user info screen displays details about the individual user such as the number of posts, gender, country, etc. Initially, the user screen was designed to incorporate the submit post functionality. However, it was decided that each screen should generally handle one use case. Moreover, separating the functionality allowed reusing the same template to show the user's details and other user's details by passing the username as a parameter to the router.

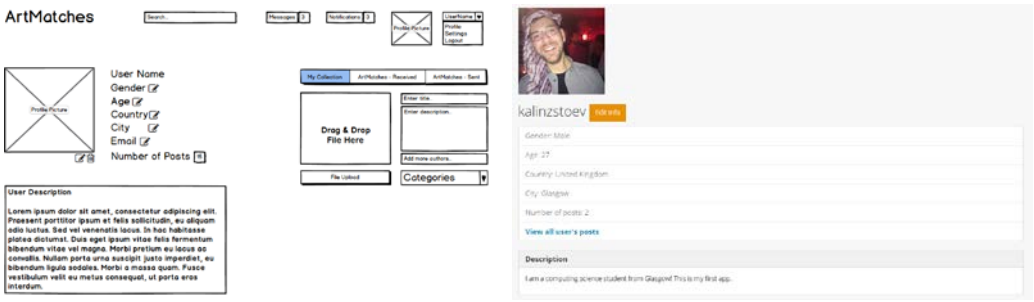

Figure 5.16 User Info initial wireframe and final design

5.4.14 View All User Posts

This screen displays all posts submitted by the particular user allowing other artists easily to see the whole work of somebody whose work they are interested in.

5.4.15 Edit User Info

Similar to the edit post screen this one allows the user to update their details (including uploading a profile picture). It does so by populating all fields initially with any existing user data and then enables the user to modify it and to save any changes.

Chapter 6 Implementation

6.1 Technology

The implementation of the technical design of ArtMatches required a multitude of technologies to meet its requirements. Each tier required a different type of technology, and thus the following section would go over which ones were chosen, why and how they were utilized to deliver the desired functionality.

6.1.1 Choosing a framework

A web application framework is a type of software which is used as a foundation for developing web applications. It typically provides common core functionality such as user authentication, data persistence, session management and templating [14]. The pros of having a framework are reducing boilerplate code, speeding up development and structuring the code around well-tested patterns. On the other hand, sometimes the WAF structure could reduce the flexibility of project. The project had to be completed within sixty days of development and thus the use of a web application was essential.

A quick search on the internet revealed a plethora of languages and frameworks to choose from. Since its user interface would run in the browser, it was predetermined that its UI would be written in HTML and styled with CSS. Additionally the only language for the browser is JavaScript, so this was a given at the beginning of the project. However, when it comes to the middle-tier, the options were numerous and very diverse. Thus, it was crucial to perform an analysis based on the produced requirements what would be the best tool to implement the job.

A quick overview of the use-cases of ArtMatches showed that most of the operations required to deliver its functionality would be data-intensive, multi-user, high traffic with no significant computation at the back end. In its essence, ArtMatches covers all the elements of SCRUD (search, create, read, update, delete) allowing multiple users to interact with the same database from each client. This was deemed achievable by any mature framework such as Django, Symfony, Ruby on Rails, Java Spring, etc. However, a closer look at some of the use cases such as chat and the fact that the application had to be real-time and provide a seamless UI implemented as a SPA started narrowing the choice for a framework.

Node.js acts as a runtime environment for JavaScript which allows it to run outside of the browser. When it comes to multiple simultaneous connections, it is the growing trend in web development to use node.js on the middle tier. It excels at handling multiple concurrent requests on a single thread by using non-blocking I/O calls [15]. On the other hand, the traditional web-serving techniques would create a new thread for each request given to the server, setting aside some RAM to handle the thread stack. As a result the traditional model hampers

the system's ability to scale for multiple simultaneous connections. In terms of creating a real-time application (such as chat), it is best if the client and the server could maintain a persistent connection allowing the server to push data to the client. In fact, this is the initial problem Node.js was created to solve allowing for less server overhead by utilizing a single-thread, event-driven server architecture to conserve resources allowing for multiple concurrent connections. This feature did fit well with the ArtMatches requirements to be real-time, and it was able to let multiple users to interact with each other at the same time without wasting resources. As a result, the right tool for the job seemed to be a Node.js based web application framework.

I performed an analysis regarding which framework had to be chosen and considered seven key factors:

- Full-stack – does it provide an integrated solution to application development for each tier?
- Documentation – was it well documented?
- Maturity – was it mature enough?
- Functionality – how much functionality offered right outside of the box?
- Extensions/Plugins – did it offer plugins to handle additional functionality?
- Community – did it have a supportive community that used it on a regular basis?
- Learning curve – how challenging it is to learn to get to a smooth working process and be able to deliver advanced features?

After a researching the available full-stack solutions, three frameworks emerged as favorites (Table 6.1).

Factor	**MEAN.io***	**Sails.js**	**Meteor.js**
Documentation	4	3	4
Maturity	5	4	3
Functionality	2	4	4
Extensions	4	3	4
Community	5	3	5
Learning curve	3	3	5
Total	**23**	**20**	**25**

*(MEAN stands for MongoDB, Express.js, Angulgar.js, Node.js)

Table 6.1 Comparison between Node.js based frameworks

Each framework was given between 0 and 5 points for each factor, and Meteor.js emerged as the best choice with a total score of 25. MEAN.io came very close, but one of the significant problems it had, was that it's not exactly a framework. In fact, it is an approach comprised of four different technologies that are not integrated together, and this adds additional time to wire up all the various components of the system in order to let it function properly. This resulted in a steeper learning curve, and since ArtMatches had to be developed in a very short span of time, it was discarded as a solution. Finally, Sails.js scored poorly on three crucial factors: documentation, community, and learning curve. This implied it would take longer to develop ArtMatches and Sails.js was abandoned as well.

6.1.2 Meteor.js Overview

Meteor is a complete full-stack, open-source web application framework which utilizes JavaScript both on the front and the back-end.

The way the framework is built is to ensure that development could focus more on the application logic instead on making all components from different tiers work together. Underneath its surface, Meteor handles three types of requests: Static Files, DDP Messages and HTTP Requests [16]. DDP is the protocol that Meteor uses to communicate between the client and the server. All client-side subscription data, method calls, and MongoDB operations are communicated as DDP messages.

Examined in detail, DDP is a protocol based on JSON, and Meteor's current implementation is based on WebSockets and SocksJS. WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server[17]. SocksJS is a WebSockets emulation transport, which can be used when WebSockets is not available.

The way Meteor handles the interaction between the client and server is by using both HTTP and DDP and thus has two separate servers (Figure 6.1).

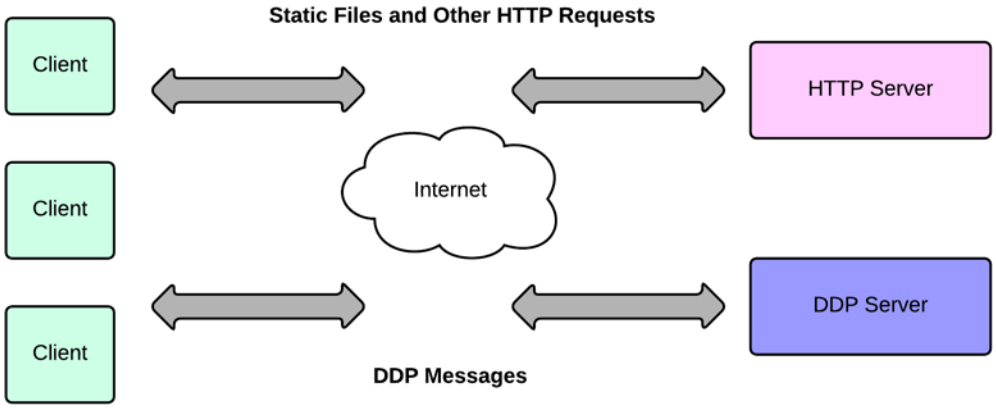

Figure 6.1 Meteor client – server interaction [16]

The main two purposes of DDP are to handle RPC and to manage data.

With an RPC, the client could invoke a method on the server and get a result back. DDP extends that feature and also notifies the caller when all write methods have been reflected to all connected clients

Managing data is done with DDP by ensuring that any time the original data source the client is subscribed to, is modified (e.g. a collection), the client is notified with a particular type of notification (added, changed, removed). This ensures that whatever the client sees as data from the database is actually real-time and up to date [16].

6.1.3 Thick Client

ArtMatches was developed as a single page application and that required writing more code that was going to run on the client. This allowed for an instant response on the client and the sought-after seamless user experience but also increased the size of the client that had to be downloaded initially. Meteor.js uses a database on the client (MiniMongo) which synchronizes with the main database and allows for instant feedback to the user and facilitates the seamless UI experience.

6.1.3.1 MiniMongo

MiniMongo is a simplified pure JavaScript implementation of MongoDB, which allows for temporarily saving data on the client (similar to a cache). This caching mechanism allows storing small subsets of the real database on the client and having instantaneous access to the data which allows for a reactive, real-time experience. MiniMongo allows what the Meteor developer group call "latency compensation." A good example would be the use case of inserting a post:

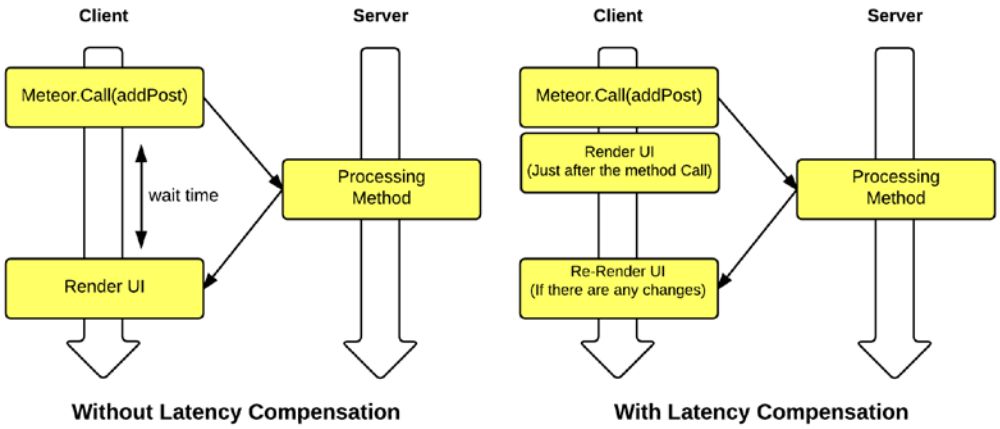

Figure 6.2 Latency compensation example [16]

The benefit of latency compensation is that it provides instant feedback to the user when performing an action. That being said, the server reserves the right to reject the change as both the mini-database and the main database are synched, and the server-side always has the last word. For example, if the user tries to insert a post, it would be visible instantly. However, if the server rejects the insert (e.g. throws an error) the change will be revoked and the UI re-rendered.

6.1.3.2 Blaze

Blaze is the front-end library in Meteor.js, which helps with developing real-time updating user interfaces. It performs the same role as popular JavaScript frameworks such as Angular, Ember, React, Polymer, etc. The benefit of using Blaze for this project was that it was created specifically for Meteor and thus integrated nicely with the rest of the application's technologies [18].

6.1.3.3 Spacebars

Meteor.js uses a template language called Spacebars which in turn is inspired by the template language Handlebars. The main difference between the two is that Spacebars incorporates reactivity. A simple example of the Spacebars syntax would be:

```
<template name="postsList">
  <div class="posts">
    {{#each posts}}
      {{> postItem}}
    {{/each}}
  </div>
</template>
```

Code 6.1 A spacebars template

This particular Spacebars HTML-like syntax tells the app to display for each post (data context) display the postItem template (the tile like representation at the homepage).

6.1.3.4 JQuery

JQuery is a JavaScript library that provides well-made and tested solutions to common problems in HTML client-side scripting. It was primarily used to select and modify elements from the DOM.

6.1.3.5 Underscore

Underscore is a utility library for JavaScript, that provides useful functions such as _.extend() which was used a lot in the development of the post creation for extending the properties of an object.

6.1.3.6 Moment.js

Moment.js is a JavaScript library that allows easy parsing, validating and manipulating and displaying time data. It was primarily used for formatting timestamp dates on the client side to make them more user-friendly (e.g. removing time-zone details to improve readability).

6.2 Middleware

6.2.1 Meteor Methods

One of the peculiarities of Meteor is that database operations could be called directly from the client. Any modification on the client-side database will try to

synchronize with the main database and if there are no rules stated it would do so automatically. However, this creates problems with security and input validation, and that's why modifications could be restricted using allow and deny rules on the server. If setting is to be avoided one of the best ways to handle, database modifications is by doing a remote procedure call from the client. The client calls a function (method) on the server (code 6.2) and passes it a parameter (e.g. a post object) which ensures that whatever operation will be performed on the database it will be controlled by the server.

```
Meteor.call('postTextInsert', post, function(error, result){
            if (error){ return throwError(error.reason);}
});
```

Code 6.2 Meteor method call for post insert

In this particular snippet, which runs on the client, the postTextInsert method is called passing it a post object and anonymous callback function which returns either an error or a result (the id of the inserted post).

6.2.2 Iron Router

Iron Router is a community-developed package that is one of the best solutions for routing between the different templates in a Meteor application. It has a very flexible functionality allowing the developer to specify a URL associated with the route. The route could take parameters or queries, then receives a name, and it automatically gets attached to the template with the same name. Moreover, Iron Router provides different hooks like waitOn that could be performed before the routing process is finished, and finally the package allows specifying the data context that is passed to the template.

```
Router.route('/user/:username', {
    name: 'userInfo',
    waitOn: function () {
        return [
            Meteor.subscribe('allUserPostsCount',
this.params.username),
            Meteor.subscribe('userData', this.params.username)
        ];
    },
    data: function(){
      returnMeteor.users.findOne(
     {username:this.params.username});
    }
});
```

Code 6.3 The route for the userInfo template

The above snippet takes a compulsory username parameter in the URL and then before it routes to the template "userInfo" it subscribes to the publications allUserPostsCount and userData. Next, it passes the user object from the user collection to the template as data context with all of its publicly available information without passing any private data.

6.3 Database

This section would first have an overview of what document-oriented databases are and then describe briefly how they differ from relational databases. The database of choice for this particular project was MongoDB because this was the only database Meteor officially supported. Additionally, having the database itself based on JSON was very convenient because all data was passed between each tier using the same data format without the need of any additional serialization.

6.3.1 MongoDB and Document-Oriented Databases

MongoDB is an open-source document-oriented database that makes storing and retrieving of data in certain applications faster and more flexible. It is a non-relational database, and its primary way of storing data is in collections (similar to SQL tables). The collections contain documents (similar to SQL rows) with a number of fields (similar to SQL columns). MongoDB, in particular, stores its documents in a JSON-like format called BSON. For example, in SQL if we would like to store a list of people – their names and gender we would create a table (Persons) and insert our record like this:

Id (Primary key)	**First_Name**	**Last_Name**	**Gender**
1234	Sally	Jones	Female

Table 6.2 SQL table example

In MongoDB the same could be done with the following - create a collection "Persons" and insert a document:

```
{
  Id: 1234,
  firstName: Sally,
  lastName: Jones,
  gender: Female
}
```

Code 6.3 BSON representation example

A significant benefit of MongoDB is that the documents in a collection are not required to be heterogeneous. This proved very useful when implementing the posts collection. It allowed posts to be stored in the same collection while having different fields (e.g. the visual posts would have an array of image id's while a written post would have a string with HTML in it).

MongoDB has a flexible schema design that differs from relational databases, and it could be easily changed throughout development. However, an initial schema design was created right from the start in the development of

ArtMatches based on its entity relationship diagram. This facilitated development and helped with consistency during the implementation.

One of the main decisions that differed from relational databases was whether to embed in the application itself or to reference [19]. For example, each comment for a post could have been embedded in a post as an array of objects. However, this would have made a subsequent lookup of comments (e.g. for modification) way slower than if each comment had been created as its own document and held a reference to the id of the original post. As a result, all comments were separated in their own collection, which was implemented similarly to the user notifications, submissions, and messages. On the other hand, sometimes embedding could be a more efficient solution. A good use case will be if no heavy update operations are expected on the data such as in the case of the visual post that had to contain multiple images. All images were stored in a separate collection. Additionally, the visual post had an array of image id's which allowed a very easy fast way to both retrieve images and reduce the amount of lookups in the image collection (e.g. searching for unique id, instead of post id).

6.3.2 Collections

There are 12 collections in the application that contain all documents in the database. The complexity of the documents in each collection varies and depends on the entity they represent. For example, the "messages" collection is very simple having several fields of which some are denormalized like the author's name (MongoDB is non-relational, and sometimes it is preferable to do denormalized data versus joins). On the other hand, a crucial collection related to the main functionality like "posts" had a way more complicated model shown in Code 6.4.

```
{
    "_id" : "AipC5TrE7KKEMA8T4",
    "postType" : "audio",
    "title" : "Michael Jackson - Billie Jean",
    "embed" : false,
    "thumbnail" : "BekiA2pYF4NRiSGPu",
    "content" : "Sz4KXRxig5LMYtNsc",
    "description" : "A timeless classic from the king of pop.",
    "category" : "pop",
    "tags" : [
        "pop",
        "moonwalk",
        "michael jackson",
        "king of pop"
    ],
    "isContentPresent" : true,
    "userId" : "QA5N7SdE5Gt53fprS",
    "author" : "kalinzstoev",
    "submitted" : ISODate("2015-09-08T20:26:52.120Z"),
    "commentsCount" : 0,
    "likers" : [],
    "likes" : 0
}
```

Code 6.4 Example of a data model of a post document

Notable is that each post in the collection could be of a different "postType" resulting in a slightly varying model. Additionally, instead of containing any media files in the document itself, the corresponding fields contain the id's of the files. Whenever the template is rendered the system performs a lookup in the "audios", "images", "thumbnails" collections and retrieves the files' URLs pointing the real files stored on Amazon S3.

6.3.3 Publications and Subscriptions

For any data to be accessible on the client side in Meteor and respectively in ArtMatches a collection has to be published and subscribed to. A publication instructs the server what kind of data is to be published to the client (e.g. it would not be prudent to publish any sensitive data to the client). On the other hand a subscription would actually decide what part of that published data should be available in the client-side collection at any point and time (e.g. it would be an overkill to have all posts available if we only need one).

Code 6.5 shows an example of how the collections in ArtMatches are published to the client. In this particular snippet the publication takes a parameter postId, next, uses the audit package to check that it is a string. Finally, it returns all comments that have the value of postId in their own field with the same name.

```
Meteor.publish('comments', function(postId) {
check(postId, String);
return Comments.find({postId: postId});
});
```

Code 6.5 Publication of the comments collection for a single post

After a collection is published in order for the client to access, it has to tell the server which subset of the data it needs to use. In this particular code snippet (Code 6.6), the subscription tells the server to return all comments with a particular post id. This particular subscription is done during the process of routing (every time the application screen changes to the View Post screen) , and thus it takes the post id from the template URL as a parameter.

```
Meteor.subscribe('comments', this.params._id);
```

Code 6.6 Subscription to the comments collection for a single post

6.4 Code Design and Structure

ArtMatches was built using a web application framework, and this automatically resulted in having certain software design decisions imposed on the project by default – one of the typical consequences of using a framework. Meteor is quite forgiving in this regard, but there were several rules that had to be followed. First, Meteor uses what's called an MV* architecture – an architecture that could be turned into a Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) architecture depending on the project and the developer. In this particular project, a MVC-like architecture was chosen primarily because it was an architecture I was much more familiar with, and it allowed a good design, offering an acceptable level of separation of concerns and extensibility. There is a slight difference between a standard MVC and the one employed in this project (Figure 6.3). The difference is due to the second model (MiniMongo database)

provided on the client and the thick client where there was more logic moved away from the server. As a result, there are two MVCs – one on the client and one on the server – interacting with one single source of truth model (the main MongoDB database).

In terms of code structure, ArtMatches was developed according to the Meteor specification [18]. It follows the rule that all code that only has to run on the client is put in a folder "client". Next, all code that runs only on the server is put in a folder "server". Then, all code that could run on both (allowing code reuse) is located in a folder "lib" and finally all tests are organized in a folder "tests". Additionally all code was structured around each of the 15 different screens discussed in the design section. For example, on the client, the homepage template (view) would be in postsList.html and all of its helpers (controller) would be in postsList.js. Any other code not directly related to the templates was structured around functionality, for example, all routing operations were put in "router.js", all publications in "publications.js", etc. Finally, all CSS code was separated into two external files. Thess CSS files are one for the BootSwatch theme and one for any CSS classes made specifically for this particular project. There was no inline CSS written in the templates and thus the project followed best practice encouraging future maintainability.

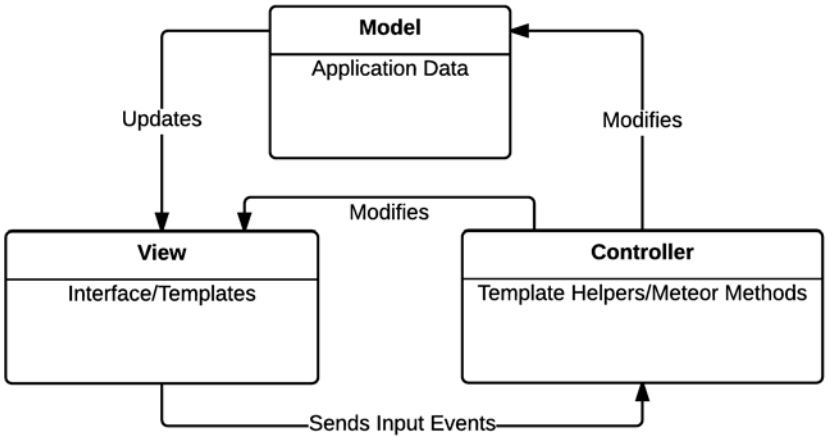

Figure 6.3 ArtMatches MVC architecture

6.5 Functionality

The dissertation so far has covered the design, architecture and the technology with which ArtMatches was built. The next few subsections will look into its core functionality and some of the significant challenges faced during development.

6.5.1 File Storing and Uploading

The core functionality of ArtMatches was to allow artists to publish different artistic mediums on the web that included images and audio files. Unlike text that is not very data heavy, files required a different solution accommodating for their size and subsequent retrieval. One of the options for file storing was just to store each file directly in the MongoDB database that allowed for up 16MB. The problem with this approach was that it was not very flexible. If in the future the

application decided to use larger files (e.g. videos) it would need full refactoring, and that could have proven time-consuming and disrupting to its users. As a result, it was chosen all files (heavy data) to be stored in an external database. After quick research, the Amazon S3 proved to be the perfect solution with free tier offering 5GB of free storage for a year.

In order to use Amazon S3, a new account was created with a designated bucket for the project. Next, a permission policy was created, and a new user with its credentials and access keys was assigned to the bucket. Storing on S3 presented the first significant challenge in the project of how to handle the file uploading from the app to the external database. A thorough overview of the Meteor documentation showed that there is no official package released by the group for this type of functionality. Therefore, either a custom solution had to be made from scratch or the project had to use an external community-made package.

The problem with community-made packages is that they are sometimes poorly documented, not thoroughly tested and not well maintained. Nevertheless, after analyzing several different packages two resulted as the primary options Lepozepo:S3 [20] and CollectionFS (CFS) [21] – a Meteor implementation of the GridFS functionality of MongoDB. CFS had much more functionality but was very briefly documented, and it didn't cover much on how to set it up. On the other hand Lepozepo:S3 was very straightforward and well-documented albeit with very limited functionality. After a several-day struggle with CFS, the easier package was chosen for implementation. However, further down development using the Lepozepo:S3, the file uploading became a problem due to the very limited functionality of the package and its difficult to extend structure. A full refactoring of the file uploading was required, and the only viable solution was CFS with its complex setup and lack of documentation. The extra time needed for the refactoring posed a serious risk towards the project. Fortunately, two days later a comprehensive tutorial was released in the Meteor community addressing the exact problem of setting up CFS and finally the implementation of the file uploading could be delivered as expected.

One of the big benefits of CFS which required additional setup was the integrated use of ImageMagic [22] – a software suite that offers image modification functionality. ImageMagick was extremely useful because it allowed the creation of thumbnails. ArtMatches is a data heavy application, and a potential lack of thumbnails would have resulted in very slow loads for the home page where a lot of the content is comprised of images.

The current implementation of CFS in ArtMatches uses an S3 adapter that first uploads the files on the server and then with image manipulation passes them to S3 using streams for both operations.

6.5.2 Text Editing

ArtMatches offers the ability to artists to post written art (poems, short stories, epitaphs) on the application. However, if a simple text area was used it wouldn't have been able to support formatted text such as paragraphs, indentations, and different font styles. Consequently, an extended functionality was needed where either the user should be allowed to add HTML formatting to their text via direct input or they would use a WYSIWYG editor. One of the core aims of the application was to provide a seamless experience for the user without additional

training and, therefore, the latter option was taken. Similar to the file uploading an external community package had to be used but, luckily, this time there was an excellently documented and developed option – Summernote [23]. This particular JavaScript library allowed an entirely customizable text editor which was well tested and widely used. After some tweaks to remove certain unnecessary functionality, such as adding tables, the library was successfully integrated with the rest of the interface. Additionally, it shared the same look with the front-end Bootstrap framework and allowed the editor to blend smoothly with the remainder of the design (Figure 6.4).

Figure 6.4 Summernote implementation in ArtMatches

6.5.3 Post Submission and Editing

All post submissions were done using forms on the client that would call methods on the server to handle the database insertion. A significant challenge was validating all user input and handling all errors. There are several options to do so, and writing fully validated forms is a very laborious task to implement from scratch. Therefore, the initial idea was to use the community package AutoForm [24] which would handle most of the form's validation automatically. However, similar to the CollectionFS package it was poorly documented and relatively inflexible for any custom cases (such as handling the submission of multiple images) and thus after three days of development it was discarded due to being hard to fit with the rest of the app. As a result, all form submission, validation and editing for the whole application was implemented from scratch.

Implementing all forms took a significant time of development, but it was essential to be done in a consistent and robust way. Consequently, ArtMatches makes sure that users can't submit posts without all required fields and checks all field types when passed to the server. Next it makes sure that only certain file types could be uploaded. Finally, it ensures only certain fields could be edited by the appropriate users and thus guarantees that all the data in the application meets the standards for confidentiality and integrity expected from a production-ready application. The following code snippet (Code 6.7) illustrates the postTextInsert method whose most important job is to perform a thorough input validation. This was essential because JavaScript is a dynamically typed language, and MongoDB doesn't enforce a database schema by default. Additionally, the post-collection had distinct allow and deny rules to make sure that all data was secure and available only to the users it was intended to.

```
postTextInsert: function (postAttributes) {
        check(this.userId, String);
        check(postAttributes, {
            postType: String,
            title: String,
            thumbnail: String,
            description: String,
            category: String,
```

```
            tags: [String],
            text: String
        });
        var cleanText = UniHTML.purify(postAttributes.text);
        postAttributes.text = cleanText;
        var errors = validateTextPost(postAttributes);
        if (errors.title || errors.category || errors.text)
            throw new Meteor.Error('invalid-post', "You must fill
            all required fields");
        var user = Meteor.user();
        var post = _.extend(postAttributes, {
            userId: user._id,
            author: user.username,
            submitted: new Date(),
            commentsCount: 0,
            likers: [],
            likes: 0
        });
        var postId = Posts.insert(post);
        return {
            _id: postId
        };
    },
```

Code 6.7 postTextInsert Method implementation in ArtMatches

6.5.4 Notifications

All notifications in ArtMatches are real-time, similar to the ones in Facebook. Because the rest of the application was built on the idea of instant feedback to the client in case of any changes in the database, it was relatively easy to implement this type of functionality. Currently, the application has two types of notifications - one that is submitted every time somebody comments on a user's post and another when a match has been submitted to a user's post. The implementation of this functionality benefited greatly from the document-oriented nature of MongoDB, which allowed having different types of notifications in the same collection allowing for code reuse and easier to extend for more types in the future. The real process of creating a notification was as simple as calling a method to insert a new notification every time a new comment was submitted (Code 6.8).

```
        comment._id = Comments.insert(comment);
        createCommentNotification(comment);
```

Code 6.8 Calling a method on the server in the comments.js collection to create a comment notification

6.5.5 Infinite Pagination

ArtMatches relies on having as much user-submitted content as possible to make it more interactive and engaging. One of the challenges associated with this was how would one display all of that content without overloading the client with

data. Additionally, because of Meteor's second database client it would need to subscribe only to the posts it needs.

Considering all of the above an infinite pagination functionality was implemented. It works by taking a number as a parameter from the route URL (e.g. http://www.artmatches.org/new/**16**) and subscribes to the same amount of posts. At the bottom of the page, there is a button "load more" that increments that number by 8 and re-subscribes to the posts collection also updating the router providing a new URL (e.g. http://www.artmatches.org/new/**24).** If there were no more posts to be loaded the button would not be shown on the page.

6.5.6 Chat

The chat functionality was implemented later in the development stage, and successfully leveraged the real-time capabilities of ArtMatches. It was developed by creating two separate collections one for chat rooms and one for all the messages. Each message was given the id of which room it belonged and who sent it. In terms of security, all messages and rooms were fine to be published and subscribed on the client (publicly available to everybody). The only security issue which had to be handled was to set up the appropriate permissions so that insertions could be allowed from the client only if the userId in the message was the one of the user trying to submit it in the first place (e.g. without these rules, users could send messages on the behalf of others).

6.6 Security

Security was taken seriously when implementing ArtMatches especially because of recent high-profile user data leaks, such as Ashley Madison in 2015, which could potentially ruin the reputation of an application and dramatically reduce its user base.

The first line of security of the application was setting up the appropriate allow and deny rules for each database operation and making sure that every input was validated against its expected type. Second, All HTML-formatted text (all written posts) was put through an HTML purifier that removed any "<script> </script>" HTML tags which could contain malicious code that would allow for cross site scripting attacks (XSS). Third an extra layer of security was introduced by utilizing Content Security Policy (CSP) which prevents XSS attacks by specifying the domains that the browser would consider as valid sources for executable scripts [25]. The CSP was defined by using the browser-policy community package, and all valid domains were specified in a configuration file on the server.

At its current stage of development, ArtMatches has two security issues that should be addressed in the future. First, at the moment there is no working user registration verification which makes the app vulnerable to spambots. This threat would be dealt with in the future by implementing an email verification functionality that would allow only verified users to do any database modification operations such as submitting posts, likes, comments, and matches. Second, ArtMatches is vulnerable to Denial of Service (DoS) attacks, and at this point doesn not have an application layer protection. However, this would be addressed in the future by implementing a server firewall by using the Sikka package [26]

that could mitigate the risk of the server going down due to too many unexpected connections. The package achieves that by requiring the IP generating too many connections to go through a ReCaptcha to proceed with using the application again. Although this would not eliminate the risk of DoS attacks, it would help with mitigating against the most basic cases especially because WebSockets-based applications are particularly vulnerable to this types of attacks.

6.7 Testing Strategy

Software testing is an integral part of delivering quality software products that meet user expectations [9]. The testing of ArtMatches was approached in two ways. The primary testing was done carrying out manual user acceptance and integration testing by the developer after every significant change or new implementation in the software. The functional requirements and use cases gathered in the planning stage were used to design appropriate test cases to check whether the system was actually doing what it was supposed to. This type of testing was useful in the early stages of development but as the application complexity grew the need for automated testing was apparent.

To perform automated integration and unit tests the application used the official Meteor testing framework Velocity, which supports the syntax of the popular JavaScript testing framework Jasmine. Jasmine was both used for unit and integration tests by using its simple syntax to create test suites (Code 6.9).

```
describe("User login and security test", function() {
    it("should not show submit-post link to anonymous user",
function () {
        var div = document.createElement("DIV");
        var comp = UI.render(Template.header);
        UI.insert(comp, div);
        expect($(div).find("#submit-post")[0]).not.toBeDefined();
    });
```

Code 6.9 An integration test example

The current automated test coverage of ArtMatches is relatively small. The reason for its size is the very short span of time in which the application had to be developed and tested. However, comprehensive automated test coverage would be one of the key areas of improvement in the future making sure that the application stays robust as it evolves its functionality.

Chapter 7 Evaluation

7.1 Evaluation Design

During its early stages of development, ArtMatches was evaluated primarily using heuristic evaluation that involved judging how its current UI compared to the existing well-established UIs of popular social networks such as Facebook, DeviantArt, SoundCloud, and ArtStation. The heuristic evaluation helped significantly with meeting its non-functional requirement in taking an incremental approach towards its UI and thus allowing new users to feel at home and feel less confused about its functionality. However, this approach was still very limited because it only inferred how users would interact with ArtMatches without any real data. Consequently, in the later stages of development, a comprehensive evaluation study was conducted (see Appendix B for its full questionnaire).

The application's evaluation study consisted of two iterations asking a group of 6 participants in each iteration to perform 8 tasks which were related to the core functionality of the app and then participants were asked to evaluate each task. Having two iterations for evaluation allowed an agile-like development where features would be modified based on user feedback and evaluated again, and this eventually improved the user interface usability.

The task evaluation was done by adopting the widely spread Technology Acceptance Model (TAM) which stipulates that technology adoption is driven primarily by two factors – perceived usefulness and perceived ease of use [28]. Additionally, the element of time was introduced by timing users for each task and asking them to what extent they were content with the time required to perform the task. After the users had completed all tasks, the overall application usability was evaluated based on the widely known System Usability Scale [29]. Finally, all users were asked to comment on what they liked the most/least in ArtMatches and what could be improved in the future.

Each user evaluation was carried out with me being present in the interaction between the user and the application which provided additional insights by observing the user's behavior. All participants who took part in the evaluation were briefed on what the application was but didn't receive any training prior to the evaluation.

.

7.2 Participant Profile

Before each evaluation, there was a quick questionnaire about the participants background which was related to the application. All participants were asked about their experience with other social networks (Facebook, DeviantArt, Soundcloud, ArtStation) and additionally they were asked if they are interested in art and if they have created art themselves.

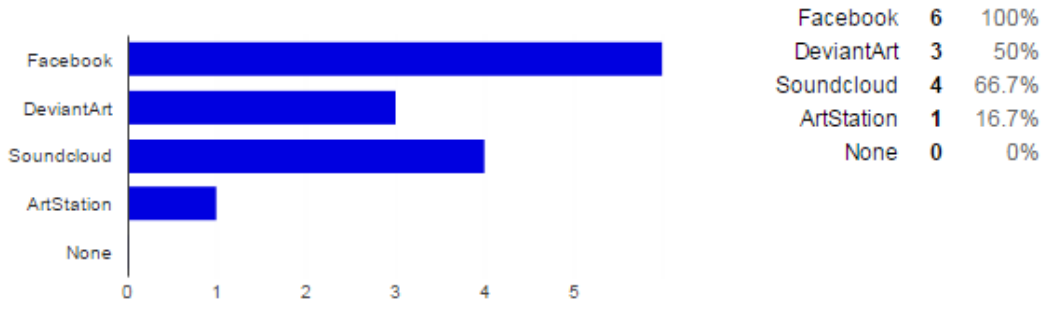

Figure 7.1 User experience with other social networks – iteration 1

In the first iteration, all participants agreed or strongly agreed that they are interested in other people's art and all declared that they have created art of their own.

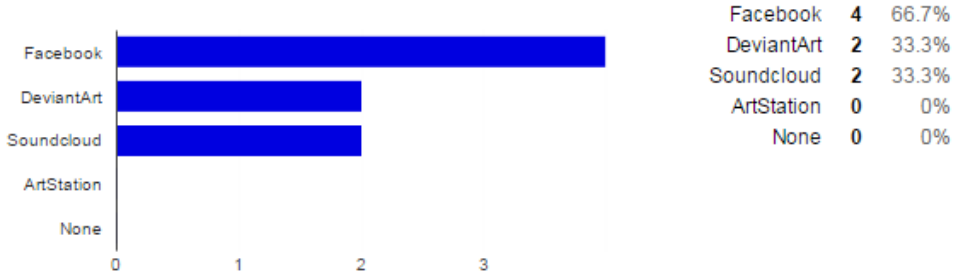

Figure 7.2 User experience with other social networks – iteration 2

In the second iteration half of the participants agreed that they had an interest in other people's art, and only one declared that they have created art of their own.

7.3 Evaluation Results

Below is a list of all the tasks that each user had to perform and then evaluate. All tasks were phrased in such a way to be realistic, actionable and devoid of any clues or describing steps. This was done in line with making the evaluation as close to the real world as possible and to make sure that the answer was not given away by the task itself [30].

Tasks:

1. Upload a new profile picture
2. Find the top 3 most liked posts which deal with visual art
3. Create a post which has a painting in it

4. Browse audio posts from the category: Pop
5. Create an audio post with an embedded song from SoundCloud
6. Submit a match to the post “Leif the Lucky”
7. Find all the posts published by the author of the post "Leif The Lucky"
8. Examine the content of a match submitted to an ArtMatch post and vote on it

In both iterations, all tasks were perceived as useful (4 and above on a five-point Likert scale) making sure that all the core functionality was something the users needed. All tasks but one were also considered easy to use, effortless and not time-consuming.

7.3.1 Iteration 1

The first iteration uncovered a serious flaw with the usability of ArtMatches, which was crucial to its functionality. The voting on an ArtMatch post emerged as a significantly more difficult task to perform than everything else. The biggest challenge users faced was finding the voting section of the ArtMatch page screen and locating the vote button for each match submission. The voting section uses tabs to display all submissions by art medium (visual, audio, written) and initially the application didn’t have any tab open by default. That made it hard for users to realize there were any submissions in this section in the first place. As a result, for iteration 2 the application always had one tab open. Additionally when there were 2 visual matches next to each other they were displayed on 6 columns each using the 12 grid Bootstrap layout reducing the amount of scrolling the user had to perform in order to get to the voting section. These measures proved to be very effective reducing the average amount of time needed to complete the task from 107 seconds to 47 seconds, and it increased its average perceived lack of effort from 5 to 8.83.

Another usability problem, the majority of users faced in iteration 1, was user profile editing. It was due to a confusing separation between one dropdown menu “My Profile” which contained a link to the user’s info and posts and another dropdown on the most right-hand side responsible for logging in/out (Figure 7.3). This was remedied for iteration 2 by merging the two dropdowns. The change resulted in decreasing the time to edit one’s profile from 37.50 to 20.50 on average and increasing the perceived time satisfaction to the maximum of 5 on a Likert scale.

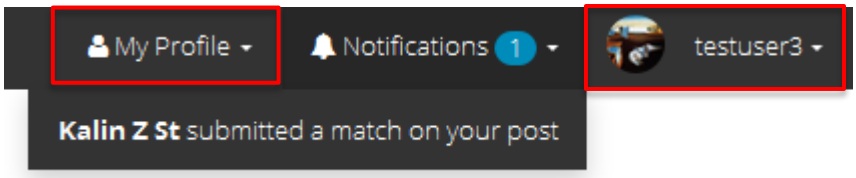

Figure 7.3 Separate dropdown menus for user profile related tasks in iteration 1

Another part of the functionality which proved to be a hurdle for all users in iteration 1 was the way category filtering for posts on the home page was implemented (Figure 7.4). Each art type in ArtMatches has different categories some of which overlap and some don’t. Initially when the categories were implemented it was assumed that the user could browse just by categories without selecting an art type. It was created intentionally to allow overlapping

categories to capture different artistic mediums in the results (e.g. one could get a photo and a song both categorized in comedy). However, this proved to be a messy solution and ended confusing the users and, therefore, the second implementation in iteration 2 required the user to first choose an art type and then to be able to select a category which made it much more straightforward. This UI modification reduced the time needed to browse posts by a category from 22.83 seconds down to 11.17 seconds and increased the perceived lack of effort to the maximum of 10.

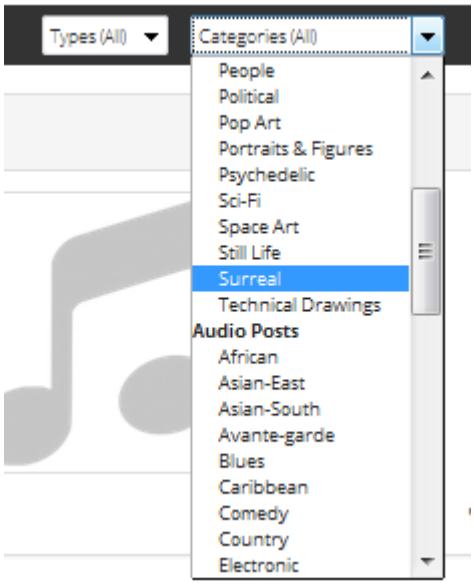

Figure 7.4 All categories showed together – iteration 1

Additionally the overall look of ArtMatches was slightly changed by increasing the font size of all links and dropdowns in the main navigation bar based on two users' feedback. Furthermore, the like button on the view post screen was changed from blue to green because some of the users thought it was a Facebook like button. That showed that the app was successfully mimicking the interface of existing social networks but was running the risk of doing so at the expense of confusing the user.

7.3.2 Iteration 2

In the second iteration, almost all tasks were perceived by the users to perform better with less effort and faster. Two tasks (3 and 5) had a weird anomaly where they took longer to complete on average compared to iteration one, but the users were more satisfied with the time it took to complete the task. This anomaly could probably be attributed to the relatively small sample of 6 people for each iteration that would allow even one outlier to skew the results in a significant way.

Task	Iteration	Perceived Lack of effort (1-10)	Ease of use (5-Likert)	Time elapsed (seconds)	Time satisfaction (5-Likert)	Perceived usefulness (5-Likert)
1	**1**	9.17	4.50	37.50	4.50	4.67
	2	10.00	4.83	20.50	5.00	4.83
2	**1**	8.50	4.33	27.33	4.50	4.83
	2	9.50	4.83	18.00	4.67	4.83
3	**1**	8.67	4.33	43.17	4.50	4.67
	2	9.67	5.00	63.00	4.83	5.00
4	**1**	8.83	4.50	22.83	4.33	4.67
	2	10.00	5.00	11.17	5.00	5.00
5	**1**	8.50	4.00	52.00	4.17	4.17
	2	8.17	3.83	69.17	4.33	4.50
6	**1**	9.00	4.17	42.67	4.33	4.67
	2	9.67	5.00	25.67	4.83	5.00
7	1	8.00	3.83	37.00	4.33	4.83
	2	8.17	3.83	34.83	4.50	4.50
8	1	5.00	2.67	107.67	2.83	4.33
	2	8.83	4.17	47.83	4.67	4.83

Table 7.1 Task evaluation results for iteration 1 and 2

The bulk of the feedback in iteration 2 did set the direction for future work for the application in terms of UI. One of the frequent suggestions from users was to utilize better the white space in the view post screen. In the current UI design, there is plenty of space to the left and to the right of the content which could include recommended suggestions and some additional functionality. Moreover, the application could use some more color as the current design looks stylish but a bit monotonous with its heavy reliance on black, white and different nuances of gray. Finally, the application should be able to handle big images better (above 2MB) which right now load relatively slowly. Moreover, since they are shown in a smaller scale, they don't really need to be that big. Addressing this issue would be relatively easy. The ImageMagick package, currently used to create thumbnails for the homepage, could also be utilized to reduce the images to a size

big enough for most screens (1600x1200px) while still retaining smaller sizes of less than 1MB.

7.4 Overall Usability

Utilizing the System Usability Scale as a tool to measure the overall usability of ArtMatches helped to gauge the cumulative effect of tweaking the interface of the individual task and also to what extent the whole application met its non-functional requirements. The scale measures the system's overall usability from 0-100 using ten questions based on a five-point Likert scale. The questions alternate in a way that they make the participant read each question carefully and mitigate the risk of the user just choosing the same option throughout the whole survey. According to the US Department of Human Services, the average score for SUS is 68 [29]. ArtMatches scored on both iterations with an above average score of 81.66 and 88.33 respectively for each. These results suggested that the application had met its non-functional requirement to be intuitive and easy to use for users who didn't have prior knowledge of the application's interface.

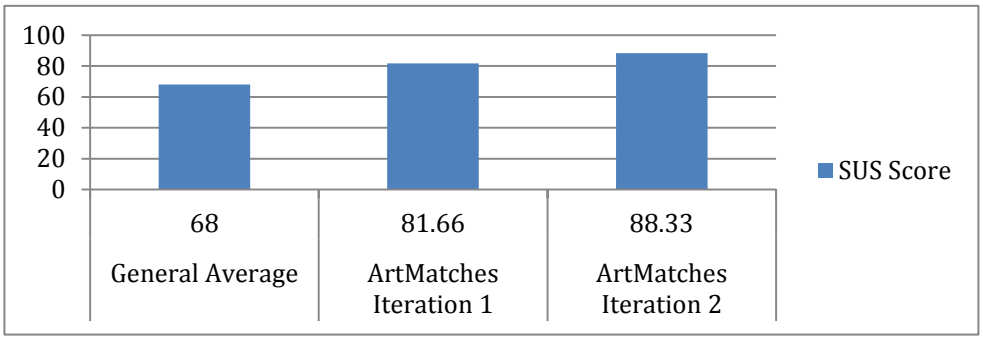

Figure 7.5 System usability scale score comparison

Finally, the application evaluation asked the participants if they found using the application to be fun. That question evaluated if ArtMatches was in line with its initial goal to help artists collaborate and due to its non-commercial nature it would need a motivation factor, such as recreation to drive its continuous use. All users agreed with an average score of 4.50 and 4.83 for iteration one and two that they found using the application fun. These results indicated that they might like to use it in the future on a continuous basis.

7.5 User Experience

One of the non-functional requirements for ArtMatches and a differentiator from existing social networks for artists was its real-time capabilities. Therefore, during its evaluation phase the application was evaluated to what extent the user could perceive it as real-time and instantaneous. According to Jacob Nielsen one of the most influential authorities on user experience research there are three significant thresholds when it comes to time perception during a user interaction [31]. All of these thresholds are based on the response time of the application.

- **100 ms** – The interaction feels to the user as instantaneous. No additional cues are needed to inform the user that a background operation is being done besides displaying the result.

- **1000 ms** – The interaction feels to the user as seamless, not interrupting their flow of thought. There is no explicit need to show any cues that any operation is carried out in the background. However, the interaction doesn't feel instantaneous anymore.

- **10000 ms** – This is the limit at which the user would switch their attention to something else. It is crucial to show a cue of some kind to the user that there is an operation being performed in the background in the range of 1000ms – 10000ms.

In order to measure some of the core functionality of ArtMatches, the Google Chrome Devtools "Timeline" feature was utilized. It allows the developer to see an event unfold from its trigger, past any computations carried out, next the final update of the DOM and finally rerendering the changed element, documenting and timing each step. The timing functionality was used to measure the response times of the functionality of ArtMatches.

The real-time functionality of the app such as sending a message in the chat, likes and comments which required an instantaneous feel currently perform at an average of 130ms to 260ms per operation. These values are outside of the instantaneous feel of 100ms and failed to meet its initial non-functional requirement. However, at they are current level they still feel seamless and, in fact, are faster than the ones of Facebook, which currently takes about 720ms to display a newly sent message and 850ms to display a new like. An instantaneous response time is not crucial for this type of functionality as long as it feels seamless. However, in the future these same response times might need to be reduced if a real-time collaboration is implemented (e.g. group drawing).

ArtMatches is a SPA that allows a quicker page load at the expense of longer initial load for the whole application. Currently, it takes on average about 1330ms to download the whole client where a spinner (Figure 7.6) is used to inform the user that application is being loaded. All further page loads are done in the range of 200-470ms per page, that successfully met the non-functional requirement for seamless experience (keeping loads under 1000ms).

There is one operation in the application – file uploading – that could take a long time to complete and could potentially go over the 10000ms threshold. Any file upload informs the user that an operation is carried out using an upload bar in the form of a GIF (Figure 7.6). One way that the user experience could be improved in the future would be through an implementation of a progress bar which reflects how much of the file has been uploaded to Amazon S3.

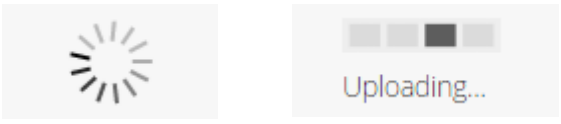

Figure 7.6 Spinner and uploading bar UI elements

Chapter 8 Conclusion

8.1 Final Product Status

The final status of ArtMatches, at the end of this dissertation, is that it has all of its core functionality (post creation, matching, etc.) developed and tested for user acceptance. Furthermore, a good amount of its supporting functionality (comments, messaging, likes, etc.) is ready to use and tested that it meets its initial requirements. However, the application is not production ready at this stage. It's current status would be best described as being in a beta stage where the product is prepared to be tested by people outside of the community/organization who developed the software. It is feature complete but does have known limitations and bugs [32]. As a result, further external testing would allow to verify the robustness and reliability of the system and potentially uncover further deficiencies/bugs that haven't been uncovered during its evaluation stage.

8.2 Shortcomings

8.2.1 Comprehensive Automated Testing

One of the areas where ArtMatches needs more work is extending its automated test coverage (both unit and integration). Comprehensive automated testing would be immensely helpful for regression testing as new changes are introduced to the system. The application is currently in its beta stage and would probably receive a significant amount of feedback from its first beta users on what could be made better. Due to the increasing size and complexity of ArtMatches any code change has the potential to affect the rest of the application in spite of having maintained a relatively acceptable amount of low coupling througout development. Thus, developing more automated tests would allow for continuous integration, faster development and eventually faster response to user feedback.

8.2.2 Security

The application needs more work related to its ability to handle user verification and protecting itself from spambots whether using email confirmations or a service similar to ReCaptcha. Furthermore, it needs more work in terms of making sure that it is well defended against any XSS attacks or other ways where attackers could try to obtain user data. Currently, all passwords are encrypted in the database but information such as email addresses is not. Future work could be done to address that all sensitive user information is protected even if potential attackers do get access to the system's database.

8.2.3 Extensibility

One of the benefits of using a web application framework was that it ensured that the code fit with a well-established structure (MVC and Meteor conventions) that provides further extensibility of ArtMatches. However, there are certain parts of the code where refactoring would be beneficial such as untangling some of the code – e.g. currently the notifications have to be triggered by comment or submit-a-match insertion methods. Moreover, because the similar functionality of the post creation for different post types, some Meteor methods could benefit from abstracting some of their common functionality to a separate method making the code more reusable.

8.3 Deployment

Currently, the application is deployed on artmatches.org and is hosted on Modulus.io – a popular service for deploying Node.js applications. The service provides an excellent solution to scalability by deploying the application to multiple mini-servers called "servos" that can be dynamically added and removed and any traffic is automatically balanced between them. The same service also hosts the application's MongoDB database offering a cloud-based, flexible storage based on the application needs which is easily extendable. Finally, the rest of the database that is situated in Amazon S3 also benefits of the flexibility of the cloud and again could easily be extended if need be beyond its current capacity of 5GB. As a result, ArtMatches has fully embraced the cloud making sure that its underlying infrastructure can meet any performance challenges as its user base grows guaranteeing the application will stay robust and scalable.

8.4 Maintenance and Evolution

Throughout the development of ArtMatches, a lot of effort was put into writing self-documenting code using long and appropriate variable and function names, structuring the code in an organized manner and supplementing with additional comments where necessary. The self-documenting code helps with the maintainability of the application and acts as its first point of reference when it comes to documentation. Furthermore, the current piece of work will serve as an additional document that highlights the most important parts of the application's architecture, technology decisions, specific functionality implementations and potential shortcomings. Finally, a good amount of the application uses external libraries to implement its functionality. Consequently, extra effort was made throughout development to choose mostly libraries that were well-documented and maintained on a frequent basis ensuring that they will be compatible and secure as the application matures in time.

In terms of evolution, the development and maintenance of ArtMatches would need to focus more on adding some of the additional functionality which was not implemented due to the dissertation's time constraints.

8.5 Future Work

The first direction for future work in the development of ArtMatches would be addressing its shortcomings and thus moving closer to an official release and a

production-ready state where it could be released in an official beta. The next step would be implementing all the "could have" and remaining "should have" functionality listed in its initial requirements (Appendix A). Some of this functionality is highlighted in the next three sections.

8.5.1 Administration

A functionality which would be crucial for a production-ready application is an administration panel. The panel could offer not just a graphical user interface to administer the database but also provide statistics dashboard, the ability to message all users, etc. Currently, ArtMatches has a very rudimentary administration panel similar to Django admin panel from the Django framework. The current implementation is based on the Houston community package [33]. A well-made administration functionality would provide the ability for non-technical people to make sure that any inappropriate content is moderated and ultimately free more time for development. The ability to moderate would require the introduction of different user roles, so each user role has the appropriate privileges and access to the database.

8.5.2 User Following

The user following functionality in existing solutions such as DeviantArt works very similarly to what Twitter has. Users could link by following each other, and all followers would be notified of any actions the user being followed has performed (e.g. submitted a new post, new match). One of the benefits for ArtMatches is that because of its extended interaction with matches, users could learn about new artists just by following their favorite artist on the network and the matches she has submitted to somebody else. Moreover, because the application is real-time, it could create an engaging experience similar to the Facebook activity log.

8.5.3 Real-time Collaboration

A tremendous potential for ArtMatches for the future would be to leverage its real-time capabilities to offer collaboration tools for artists. This collaboration could be implemented in various ways to accommodate the multitude of artistic mediums that the application allows. For example, multiple painters could sketch/draw at the same time in a canvas HTML5 element, and that would enable them to exchange quickly ideas without actually having to be in the same room. Multiple writers could be able to edit the same document in real-time which could allow them to have one person leading the story line and another editing after at the same time similar to what Google Docs offers right now.

Audio collaboration could be harder because of latency and the vast amount of bandwidth required to support it. However, assuming that the current technology improves, then musicians could jam together in real time while being away from each other. Finally, because the application offers different artistic mediums it could mix these collaborations. The application could have musicians jamming while somebody else is jotting down some lyrics where they could see them real-time in the browser offering a base for creating song demos that could be later refined.

8.6 Reflection

The development of ArtMatches was a significant learning experience, and it allowed me the opportunity to go through each part of the software development lifecycle. It was very exciting to have a self-defined vague concept and see it evolve and take shape in the course of several months including the initial proposal stage. It was very rewarding to see the project become a mature prototype where it could be tested and evaluated by users.

Choosing Meteor.js as the tool for the job was also was a very interesting experience because it exposed me to a very different web development paradigm. The novelty of the framework provided its challenges with finding documentation or well-made external packages to extend functionality. However, this also forced me to think how I could deliver the best product with the tools that I had at my disposal. Additionally, using JavaScript and a document-oriented database like MongoDB showed me that there is more than one way to implement a piece of software in terms of technology and code structure. Last but not least, it was very handy to have a single data format that would move across layers (JSON) without extra serialization.

I believe the biggest challenge I had to overcome in the development of the application was not a technological one but a process one. For example, certain design decisions or underestimating the complexity of some features resulted in several significant delays that could have impacted the successful delivery of the project. In the future, I would address these by spending more time on the initial design of the project and moreover always taking with a grain of salt my initial estimates for the implementation of any significant feature.

In conclusion, this dissertation is a comprehensive record of all the different aspects in terms of planning and execution that I had to address throughout the development of the software which resulted in a mature prototype. The initial evaluators from both iterations liked the application, and hopefully ArtMatches would grow into a production-ready app at some point in the future. All of that allowed me to gain experience with a multitude of technologies required to develop a real-time SPA, and I am very glad I had the opportunity to do so.

ArtMatches (as of September 2015) is deployed at artmatches.org, and its code is freely available at https://github.com/kalinzstoev/artmatches.

Chapter 9 References

[1] Freitas, M. (2009) *DeviantArt: Where Art Meets Application*, University of Madeira

[2] DeviantArt (2015), *About*
https://about.deviantart.com/

[3] Almila, A. Salah, A. (2013) *Flow of innovation in DeviantArt: following artists on an online social network site*, Mind & Society

[4] Wikipedia (2015) *DeviantArt*,
http://en.wikipedia.org/wiki/DeviantArt.

[5] USA Today (2013) *Who's listening to SoundCloud? 200 million,* http://www.usatoday.com/story/tech/columnist/talkingtech/2013/07/17/whos-listening-to-soundcloud-200-million/2521363/

[6] Popova, M. (2015), *Literary Jukebox*,
http://literaryjukebox.brainpickings.org/

[7] Alexa (2015), *The Top 500 Websites on the Web,*
http://www.alexa.com/topsites

[8] The Guardian 2015, Facebook*: 10 Years of Social Networking, in Numbers*, http://www.theguardian.com/news/datablog/2014/feb/04/facebook-in-numbers-statistics

[9] Sommerville, I. (2010) *Software Engineering*, Addison Wesley 9th Edition

[10] Bootswatch (2015) – *Yeti Bootstrap Theme,*
https://bootswatch.com/yeti/

[11] FontAwesome Iconic Font (2015),
https://fortawesome.github.io/Font-Awesome/

[12] Toastr – JavaScript library for non-blocking notifications (2015), https://github.com/CodeSeven/toastr

[13] Wikipedia (2015) *Single Page Application,*
https://en.wikipedia.org/wiki/Single-page_application

[14] Azzopardi, L. (2014) *Web Application Frameworks*, lecture delivered at the University of Glasgow for the Internet Technology class for MSc Software Development.

[15] Capan, T. (2013) *Why The Hell Would I Use Node.js,*
http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[16] Susuripala, A. (2014) *Meteor Explained: A Journey Into Meteor's Reactivity,* https://gumroad.com/l/meteor-explained

[17] Mozilla Developer Network (2015), *WebSockets,* https://developer.mozilla.org/en-US/docs/WebSockets

[18] Meteor Documentation (2015), *Introduction*, http://docs.meteor.com/#/basic/

[19] MongoDB Blog (2014), *Six Rules of Thumb for MongoDB Schema Design,* http://blog.mongodb.org/post/88473035333/6-rules-of-thumb-for-mongodb-schema-design-part-3

[20] Lepozepo:S3 (2015) Meteor File Upload Package, https://github.com/Lepozepo/S3

[21] CollectionFS (2015) Meteor File Upload Package, https://github.com/CollectionFS/Meteor-CollectionFS

[22] ImageMagic (2015) Image Editing Suite, http://www.imagemagick.org/script/index.php

[23] Summernote (2015) WYSIWYG Text Editor, https://github.com/summernote/summernote

[24] AutoForm (2015) Meteor Form Management Package, https://github.com/aldeed/meteor-autoform

[25] Mozilla Developer Network (2015), *Content Security Policy* https://developer.mozilla.org/en-US/docs/Web/Security/CSP

[26] Sikka (2015) A Firewall for Meteor Apps, https://atmospherejs.com/meteorhacks/sikka

[27] Jasmine (2015), Javascript Testing Framework, http://jasmine.github.io/

[28] Venkatesh, V., & Davis, F. D. (2000). *A theoretical extension of the technology acceptance model: Four longitudinal field studies.* Management science, 186-204.

[29] US Department of Human Services (2015), *System Usability Scale* http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[30] Nielsen Norman Group, *Task Usability Testing,* http://www.nngroup.com/articles/task-scenarios-usability-testing/

[31] Jacob Nielsen (2015), *Response Times: The 3 Important Limits,* http://www.nngroup.com/articles/response-times-3-important-limits/

[32] Jeff Atwood (2008), *Alpha, Beta, and Sometimes Gamma*, http://blog.codinghorror.com/alpha-beta-and-sometimes-gamma/

[33] Houston (2015), Meteor Administration Package, https://github.com/gterrono/houston

Appendix A Functional Requirements

MoSCoW Analysis

Note: All user requirements in the "must" and "should haves" marked in gray were planned to be implemented but were later abandoned due to technical reasons, concept changes or due to the tight time constraints of the project

1. **Must Haves**

- User Profile
  - Profile Creation
    - upload a profile picture
    - provide personal information
      - add username
      - choose country
      - choose city
      - choose age
      - choose gender
  - Displaying and editing existing profile
    - upload/change/remove profile picture
    - add/edit their personal description and personal information
    - posts count
  - ArtMatches – received
    - browse through matches submitted
    - see the content of each submission
    - vote on each submission
    - approve matches - This functionality was changed during development to voting on all submissions without the explicit approval of the owner of the original post.
    - hide matches(see above)
  - ArtMatches – sent
    - get a notification if a match was submitted
- User Posts
  - Create post
    - add title
    - add description
    - add content (file upload)
    - add content
    - add tags
  - Visual posts – support for multiple artworks per post
  - Audio posts – support for file audio uploads from the user

    - Written posts – support for text formatting inside the browser with no technical experience and HTML knowledge for the user
    - Edit post
        - Edit title, description, category, content, tags,
    - A user can like a post (1 per user)
    - submit an ArtMatch
        - pick from existing user posts
- User posts feed
    - Posts – latest, most liked, most discussed

- Posts Search
    - search posts by category, name, tags, author name

- Organize posts into predefined categories

- Organize posts with hashtags
    - Option to add a hashtag to a post
    - Edit existing hashtags in own posts by user
    - See existing hashtags in a post

2. Should Haves

- User Profile
    - A user can log-in and out using other social networks
        - Facebook integration
        - Google + integration
- Social networks embedding
    - SoundCloud (sharing audio)
    - YouTube (sharing audio/video)
- User Messaging
    - see all messages in a chat room for each user
    - see all messages in a chat room
    - send a message
    - create a new chat room
    - select an existing chat room
- Comments
    - add
    - remove
    - edit
    - report comment
        - Get a notification every time a user comments on one of your posts
    -
- User following
    - follow other users and be notified when they post something new
    - be followed by other users

    - have a separate post stream that displays only work by posts by users that are followed
- User posts
    - add multiple authors
    - report content
    - Count page views (every time the work is seen) – This feature was abandoned due to technical difficulties explained in the implementation

- Profile verification
    - automatically send an email upon registration with a verification link
    - wait for confirmation to verify profile and let the user login – This feature was abandoned in order to allow a quicker start for new users and due to an unreliable SMTP server which sometimes would take a while to send a verification email

3. Could Haves (none were implemented)

- A user reputation system and simple gamification (badges for popular art matches, etc.)
- Administration
    - see any posts that have been reported more than 10 times
    - remove Posts
    - suspend user profiles
    - delete user profiles
    - message all users

4. Won't (Would) Haves (none were implemented)

- Personal blog system
- User forum
- Automatic recommendations algorithm
- Real-time collaboration

Appendix B Evaluation Questionnaire

ArtMatches - User Evaluation

* Required

1. **Participant Number:** *

2. **1. I have used the following web social networks :** *
   Check all that apply.

   - [ ] Facebook
   - [ ] DeviantArt
   - [ ] Soundcloud
   - [ ] ArtStation
   - [ ] None

3. **2. I regularly spend time experiencing the art of others.** *
   For example: listening to music, looking at photography or paintings, reading poems, novels and other kinds of literature.
   Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

4. **3. I create or have created art of my own:** *
   For example: painting, photography, writing, music, etc.
   Mark only one oval.

   - ◯ Yes
   - ◯ No

5. **4. If you have answered yes to question 3 please specify below:**

   .....

Task 1: Upload a new profile picture

6. **1. How easy was for you to complete the task?** *
   Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

7. **2. I was satisfied with the ease for completing the task.** *
   Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

8. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

9. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. **Time Elapsed (Seconds):** *

.....

Task 2: Find the top 3 most liked posts which deal with visual art

11. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

12. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

14. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

15. **Time Elapsed (Seconds):** *

.....

Task 3: Create a post which has a painting on it

16. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

17. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

18. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

19. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

20. **Time Elapsed (Seconds):** *

.....

Task 4: Browse audio posts from the category: Pop

21. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

22. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

23. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

24. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

25. **Time Elapsed (Seconds):** *

.....

Task 5: Create an audio post with an embedded song from soundcloud

26. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

27. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

28. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

29. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

30. **Time Elapsed (Seconds):** *

.....

Task 6: Submit a match to the post “Leif the Lucky”

31. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

32. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

33. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

34. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

35. **Time Elapsed (Seconds):** *

.....

Task 7: Find all the posts published by the author of the post "Leif The Lucky"

36. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

37. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

38. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

39. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

40. **Time Elapsed (Seconds):** *

.....

Task 8: Examine the content of a match submitted to an ArtMatch

post and vote on it

41. **1. How easy was for you to complete the task?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very hard to do	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very easy to do

42. **2. I was satisfied with the ease for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

43. **3. I was satisfied with the time it took for completing the task.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

44. **4. I believe performing this task would be useful for me when using the application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

45. **Time Elapsed (Seconds):** *

.....

System Usability Evaluation

Please express to what extent you agree with the following statements:

46. **1. I think that I would like to use this application frequently.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

47. **2. I found the application unnecessarily complex.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

48. **3. I thought the application was easy to use.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

49. **4. I think that I would need the support of a technical person to be able to use this application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

50. **5. I found the various functions in this application were well integrated.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

System Usability Evaluation

51. **6. I thought there was too much inconsistency in this application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

52. **7. I would imagine that most people would learn to use this application very quickly.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

53. **8. I found the application very cumbersome to use.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

54. **9. I felt very confident using the application.**
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

55. **10. I needed to learn a lot of things before I could get going with this application.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

Final Feedback

56. **1. I found using the application to be fun and enjoyable.** *
Mark only one oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

57. **2. What do you like most about the application?**

.....

.....

.....

.....

.....

58. **3. What do you like least about the application?**

59. **4. What would you like to see in the future to be added or improved?**

60. **5. Any other comments?**