Interactive Museum Labels

Android App for Discovering Artworks

Stephany Rhee

May 2016

Yale University
Class of 2016
seungyeon.rhee@yale.edu | srhee93@gmail.com

https://github.com/stephrhee/casso
# Table of Contents

Abstract 3

## Background 3

1. Landscape of mobile apps for art museums 3
2. Landscape of mobile apps related to art 3
3. YCBA Database 4

## Data Management 5

1. Database 5
2. Data Model 6

## User Flow 8

## Design 10

## Data Analysis 10

## Reflection 11

## Thanks 12
Abstract

The interactive museum label is an Android app that enriches visiting art exhibitions with a digital experience. It has two main functions: to serve as a digital version of artworks and museum labels; and to serve as a discovery tool for users to find related pieces using a graphical network of “Tags” of subjects, places, people, and events portrayed in pieces. As proof of concept, the app uses pieces from the Yale Center for British Art (YCBA), and can be adapted by other exhibitions in the future. This project includes a Firebase database, an Android app, and data analysis of the Tags. This report also includes descriptions of key product and interface design choices that were considered to make the app a desirable product for museumgoers.

Background

1. Landscape of mobile apps for art museums

Major art museums each have their own app, and the feature sets vary widely across the apps. Mobile apps for the three most visited art museums in the United States as reported by the The Art Newspaper in 2014 are: The Met (Metropolitan of Museum Art, New York), Your Art (National Gallery of Art, Washington D.C.), and MoMA (The Museum of Modern Art, New York). The first two exist on both iOS and Android, and the third exists only on iOS. Features across the three apps include current exhibition highlights, audio guides for tours, and ability to save your favorite pieces.

2. Landscape of mobile apps related to art

Very few apps exist that allow users to search for artworks from a universal database and gain useful information on them. On the other hand, other categories of creative content have robust online communities and apps and websites; to draw analogies to other forms of what may be considered “art”, there isn’t a “YouTube” (video-sharing app), “Spotify” (music-sharing app),

---

“Dribbble” (website for sharing designs), or “Rap Genius” (website for crowd-sourced annotations of lyrics) for artworks. Reasons for this may include:

- There is no universal digital database of artwork images. Current efforts in this area include the International Image Interoperability Framework (IIIF), which is a group of institutions and companies working to create databases, engines, and APIs that gather and standardize images of cultural works. YCBA is a member of the IIIF.

- Enjoying art is still an intrinsically physical experience. The experience of creating and sharing home-made content is naturally a digital one, which is why platforms like YouTube and Dribbble may have such robust communities. It goes without saying that music discovery, a digital experience, also lives in a very mature ecosystem of apps and websites.

The most popular app on the Google Play app store for searching and learning about art seems to be Artbit\(^5\). Artbit is under the Lifestyle category (there is no Art category in the Google Play app store as of May 2016). Artbit prompts users to search for artworks by taking a photo of it, performs image recognition technology to detect the piece, then gives the user the following information: basic details about the piece such as title, artist, year, and material; a “Did You Know” trivia such as “Kandinsky was a leader in avant-garde art, one of the founders of pure abstraction in painting in the early 20th century”; and a “See Also” section of other pieces by the same artist. While the app has a clean and colorful interface along with basic information about artworks, it doesn’t allow users to discover related pieces by other artists; perhaps it is most useful for detecting a piece of art that the user cannot identify, but not for discovering new artworks.

3. YCBA database

The YCBA has a database of every piece in their collection. Each piece has basic details such as title, artist, and year, images of various sizes ranging from thumbnail to originals resolution of 3000px, and a list of subjects, places, people, and events portrayed in the piece. The discovery feature in the interactive museum label uses these Tags of subjects, places, people, and events to suggest other pieces closely related to the piece that the user is currently looking at. To users, the closest concept to Tags they are familiar with would be what you commonly call Hashtags on a social media platform such as Instagram or Facebook. The Tags have valuable information and potential that other apps such as Artbit do not have; apps that don’t know what topics are portrayed

---

in the piece can only suggest other pieces by the artist, but having this information means the user can discover every piece in the collection that is related to “portrait”, “nudes”, “marine art”, or “India”, as an example. The YCBA’s current collection on display as of May 2016 has 444 pieces and 2035 Tags. The list of 444 pieces were gotten from Emmanuelle Delmas-Glass, the Collections Data Manager at the YCBA. There is currently no running list online of current works on display; the list of current pieces on display has to be updated in the Firebase database (see Database under Data Management) whenever the collection on display changes.

Data Management

1. Database

The YCBA data (see YCBA database under Background) can be fetched in two ways: fetch XML endpoints using Java, Python, or any other language of your choice, or fetch Linked Open Data endpoints using SPARQL queries. The interactive museum label fetches the XML endpoints with in-app queries in Java; this was an intentional choice given that I had no background with SPARQL.

Each XML endpoint contains data for one artwork and is queried using the object id number of that piece. Initially, the app fetched an XML endpoint for each time an artwork needed to be rendered in the app. This quickly proved to be expensive, especially for rendering related artworks; for example, the Tag “woman” has 82 related pieces, and this required making 82 calls to the YCBA database. Performance was a top priority in developing the app to create the smoothest experience possible for the user. The YCBA data now lives in a Firebase—a realtime JSON database and backend service—repository so that one fetch at the beginning of an app lifecycle gets a hashmap of (key, value) pairs of <String, SimpleTag>, where the key is a string representation of a Tag (e.g. “woman”), and the value is an instance of a SimpleTag class. The SimpleTag class has a class variable of List<SimpleArtwork> of the related artworks for this Tag (e.g. the 82 pieces related to “woman”). An instance of the SimpleArtwork class contains a URL for the image, which is used to render the piece. More detail on the various class model can be found in the next subsection.

---

6 The SimpleTag is to differentiate the class from the another class called Tag. They represent the same data object, but SimpleTag has fewer class variables than a Tag.

7 Same reason as footnote 6.
The initial method of fetching an XML endpoint each time an artwork is rendered has a runtime of \(O(n_t)\), where \(n_t\) is the size of the total set of Tags in the database. Each piece in the current collection has an average of 16 tags (see Data Analysis). Assuming the number of Tags per piece grows linearly with the total set of Tags, the number of Tags per piece grows with \(O(n_t)\). This is not scalable. Making 82 calls to the YCBA database, parsing the 82 XML endpoints, and downloading the 82 images for “woman”, as an example, takes around 40 seconds in a Genymotion emulator of Samsung Galaxy S5 configured with 4 processors and a base memory of 2048 MB. On the other hand, the current method of getting a list of image URLs from the hashmap has a runtime of \(O(1)\). In this current method, rendering the 82 images for “woman” costs virtually no time for getting the list of image URLs, and only requires download time for the 82 images; in the same aforementioned emulator, rendering the 82 images for “woman” takes around 15 seconds, which is a significant improvement from the initial method.

To create the Firebase database, the following steps were taken exactly once. As there no online running list of current pieces on display (see YCBA database under Background), these steps must be taken every time the YCBA changes the current collection on display.

i. Fetch XML endpoints for all 444 pieces current on display.
ii. Parse the XML endpoints, and store each piece as an instance of the Artwork class (see Data Model under Data Management).
iii. Traversing the list of 444 Artworks, create a hashmap of (key, value) pairs of <String, SimpleTag>\(^8\), where the key is a string representation of a Tag (e.g. “woman”), and the value is an instance of a SimpleTag class.
iii. Push the hashmap to Firebase.

The hashmap is fetched at the beginning of each app lifecycle and is used throughout the whole app lifecycle.

2. Data Model

The two top-level data objects are Java classes called Artwork and SimpleTag. JSON objects are generated from the Java classes and stored in Firebase, and when fetched back to the client-side, generate back as Java classes.

\(^8\) The SimpleTag is to differentiate the class from the another class called Tag. They represent the same data object, but SimpleTag has fewer class variables than a Tag.
The Artwork class contains all the information found in each XML endpoint in the YCBA database. In Firebase, Artwork instances are stored as a hashmap of (key, value) pairs of <Integer, Artwork>, where the key is an integer representation of the object id, and the value is an instance of the Artwork class. The hashmap looks like following:

```
"artworks": {
    "1000": {
        "mArtist": "Philip Reinagle",
        "mCategory": "Man-Made Object",
        "mClassification": "Painting",
        "mEndYear": "1800",
        "mGenre": "still life",
        "mHighResImage_url": "http://deliver.odai.yale.edu...",
        "mId": "1000",
        "mLowResImage_url": "http://deliver.odai.yale.edu...",
        "mMaterials": {
            0: "oil paint",
            1: "canvas"
        },
        "mObjectTypes": {
            0: "painting"
        },
        "mStartYear": "1800",
        "mTags": {
            0: "blue",
            1: "botanical subject",
            2: "passion flower",
            3: "science"
        },
        "mTitle": "Blue Passion Flower, for the..."
    },
    "1001": {
        "mArtist": "Philip Reinagle",
        "mTitle": "Blue Passion Flower, for the..."
    } ...
}
```

The SimpleTag is to differentiate the class from the another class called Tag. They represent the same data object, but SimpleTag has fewer class variables than a Tag.
The SimpleTag class contains a list of SimpleArtwork instances, which is a simplified version of the Artwork class. This is the list of pieces related to this tag. In Firebase, SimpleTag instances are also stored as a hashmap (see Database under Data Management). The hashmap looks like following.

```
"tags": {
    "singers": {
        "suggestedArtworks": {
            0: {
                "id": 1007,
                "thumbUrl": "http://deliver.odai.yale.edu..."
            },
            1: {
                "id": 1008,
                "thumbUrl": "http://deliver.odai.yale.edu..."
            }
        }
    },
    "woman": {
        "suggestedArtworks": {
            ...
        }
    }
}
```

User Flow

The interactive museum label is an Android app. The landing screen (Figure 1) is a SearchActivity, which consists of two search fields and a list of search results. Users search for a piece by typing in the artist name and the piece title. The app returns, in realtime, a list of search results that match the user input (Figure 2 and Figure 3). The user must select one of the search results. Then, the piece selected is rendered in the ArtworkProfileActivity, which shows 1) basic information about the piece and the color palette of the piece (Figure 4), and 2) a “Discover Tags” section that shows a list of Tags color-coded according to frequency (Figure 5). When a user clicks on one of the Tags, a scrollable tape of related pieces slides out under the Discover Tags section (Figure 6), and clicking on one of the pieces will open a new ArtworkProfileActivity of the clicked piece.
The Tags are color-coded in order to provide even more utility to the user. Tags with the most related artworks are the darkest (e.g. costume, man, portrait, and woman in Figure 5), and Tags with the fewest related artworks are the lightest (dance, keyboard, pianoforte, rifle, sister, symbolism, and tambourine in Figure 5). Both are useful; clicking on a the “costume” Tag will let the user discover the 135 artworks with costumes, while clicking on the “sister” Tag will let the user find the 2 only pieces in the entire collection that are about sisters. The darker Tags help the
user to discover many other pieces, while the lighter tags encourage the user to learn about a small set of images that are related to each another by a very niche topic.

Design

The app interfaces with the user as a digital museum label, as museum labels are universally recognized by museumgoers. A standard museum label has the artist and title in a large font, followed by the year, object category (e.g. Man-Made Object), relevant genres (e.g. sculpture, portrait), and materials (e.g. oil canvas, plaster) in a smaller font.

The color-scheme and the typeface of the app were intentionally chosen to best mimic the design of a museum label. For the background layer and the museum label box colors, cool, subtle tones such as #FAFAFA and #CFD8DC were used. The color palette of the Tags was chosen as a set of four colors ranging from pale yellow to red—#F6EEAC, #F0D58B, #EDAF74, and #F0A28B in increasing order of darkness—to reference the meaning of color-coding the Tags; the red Tags are the warmest as they have the most number of related pieces, while the pale yellow Tags are the least warm as they have the least number of related pieces. For the global typeface of the app, Sorts Mill Goudy\(^\text{10}\) was chosen as it best resembled what is commonly found in museum labels among free Android fonts on Font Squirrel.

Data Analysis

Data Analysis of the Tags are useful in gaining more information about the Tags in the current YCBA collection. The following statistics may be useful to the YCBA in organizing future collections.

Top 30 most common Tags (with their frequencies):

1. landscape 159  
2. portrait 152  
3. costume 135  
4. United Kingdom 112  
5. man 99  
11. trees 52  
12. river 50  
13. Europe 44  
14. women 44  
15. horse 37  
21. light 31  
22. hills 31  
23. family 30  
24. animal art 30  
25. children 30

\(^\text{10}\) https://www.fontsquirrel.com/fonts/Sorts-Mill-Goudy
Reflection

The major deliverables of this project—develop an Android app that enriches the experience of visiting art museums—have been completed. The stretch goals outlined in the initial Project Proposal—using image recognition to detect pieces when seen through the phone camera—were not pursued as other features in the app came to take priority over them. Furthermore, the most important functionalities of the app were determined by what kinds of data is available in the YCBA database, which were still unknowns when the initial Project Proposal was written; the Discover Tags feature is perhaps the unique and valuable aspect of the app, although it wasn't
known at the time of the initial Project Proposal that this information was available in the YCBA database.

Thanks

I would like to thank Professor Holly Rushmeier of the Yale Computer Science Department for advising my project. She played an instrumental role in guiding various decisions related to the technology, the interface design, and product features. I would also like to thank Emmanuelle Delmas-Glass, the Collections Data Manager at the YCBA, for connecting me to the relevant technologies and resources in the YCBA, as well as giving this project clarification and support throughout the semester. Finally, I would like to credit fellow Yale Computer Science students Shona Hemmady and Vicky Tu for providing feedback and inspiration for new ideas at our weekly group meetings with Professor Rushmeier.