A mobile application for public art

Utilising smartphones and crowdsourcing to enable users to discover and learn about public art

Ole Andreas Rydland

Master of Science in Computer Science
Submission date: February 2014
Supervisor: Sobah Petersen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

A mobile application for public art - Utilizing smartphones and crowdsourcing to enable users to discover and learn about public art

Ole Andreas Rydland
oleandry@stud.ntnu.no,
NTNU - Norwegian University of Science and Technology,
Faculty of Information Technology, Mathematics and Electrical Engineering,
Department of Computer Science,
Supervisor: Sobah Abbas Petersen

February 3, 2014

Summary

This thesis follows the background research and development of an Android application meant to utilize crowdsourcing and other techniques to build a database and share knowledge about art in public places.

Oppsummering

Denne masteroppgaven flger bakgrunnsforskningen og utviklingen av en Android applikasjon som skal bruke crowdsourcing og andre teknikker for bygge en database og dele kunnskap om kunst i offentlig rom.

Abstract

This report describes the work done in a Master Thesis conducted at the Norwegian University of Science and Technology from september to february. The purpose of the project is to design and develop a proof of concept Android application that enables the public to learn about art in public places. The application is meant to use ideas from crowdsourcing and social tools to build a database of information about art in public places. It should also enable users to access this information.

The project was conducted by studying literature and related applications before starting development on the application.

This paper provides information about the research that has helped design the functionality of the application and the details of the implementation of the application.

The paper summarizes the project in relation to the goals of the thesis, and provides suggestion for further work to improve the application. In the end, the paper concludes that the idea of utilizing crowdsourcing for the goal of helping people learn about public art is a valid one, even though it also provides several challenges that has to be solved.

List of Figures

3.1 Access to mobile phones and smartphones in Norway 2010-2012. 11
3.2 The map screen in ArtAround 12
3.3 Art information screen in ArtAround 13
3.4 Adding new art in ArtAround 13
3.5 Adding a comment in ArtAround 14
3.6 Reading comments in ArtAround 14
3.7 Map with favorites enabled 15
3.8 The categories of places in foursquare 15
3.9 The best nearby list in 16
3.10 Additional search filters in foursquare 16
3.11 Map showing search results in foursquare 17
3.12 A place in foursquare 18
3.13 A list in foursquare 18
3.14 Badges in foursquare 19
3.15 Social network in foursquare 19
3.16 Adding a report in Waze 20
3.17 A user with their score and avatar in Waze 21
3.18 The user profile showing the score you have achieved 22
3.19 A task to visit a place in Waze 22
3.20 The turn by turn interface in Waze 23
3.21 Wikipedia's front page 24
3.22 Wikipedia talk page 24
3.23 Wikipedia talk page grading 25
3.24 Wikipedia article revision history 25

4.1 Architecture 30
4.2 Datastore model 33
4.3 The interaction between interface elements 35
4.4 The actionbar 36
4.5 The actionbar buttons explained 36
4.6 Illustration of the overflow functionality 36
4.7 The choose account dialog 37
4.8 Map screen after account created 38
4.9 Markers on the map screen 38

4.10 The information fragment 39
4.11 Artwork information screen. The picture is a static picture loaded locally. 40
4.12 User adds rating 41
4.13 The dialog shown when user presses create new on the actionbar 42
4.14 The empty create new artwork screen 42
4.15 The category spinner 43
4.16 Filled out new artwork screen 44
4.17 Picture added to the new artwork screen 44
4.18 Adding a new route 45
4.19 List of routes 46
4.20 Routes 47
4.21 Routes order 47

Contents

Summary iii

Oppsummering v

Abstract vii

1 Introduction 1
- 1.1 Purpose 1
- 1.2 Motivation 1
- 1.3 Research Questions 2
- 1.4 Project goals 2
- 1.5 Development methodology 3

2 Application concept 5
- 2.1 Application's goal 5
- 2.2 Functionality 5
  - 2.2.1 Add, get and edit information 5
  - 2.2.2 Routes 5
  - 2.2.3 Assisting users in finding artworks 6
- 2.3 Requirements 6
  - 2.3.1 Functional Requirements 6
  - 2.3.2 Non-Functional Requirements 7
  - 2.3.3 Availability 7

3 Research 9
- 3.1 Literature review 9
  - 3.1.1 Crowdsourcing 9
  - 3.1.2 Attracting and motivating users 10
  - 3.1.3 Learning and mobile devices 10
- 3.2 Related applications 11
  - 3.2.1 ArtAround 11
  - 3.2.2 FourSquare 14
  - 3.2.3 Waze 20
  - 3.2.4 Wikipedia 24

3.3 Previously implemented features 26
3.3.1 Data storage 26
3.3.2 Network 26
3.3.3 Android application 26

4 Implementation details 29
4.1 Architecture description 29
4.1.1 Datastore 31
4.1.2 Data transfer 33
4.1.3 Android client 34
4.2 Tools and libraries 48
4.2.1 Client 48
4.3 Back-end 49
4.3.1 Google App Engine 49
4.3.2 Google App Engine Datastore 49
4.3.3 Google Cloud Endpoints 50
4.3.4 Objectify 50

5 Further work 51
5.1 Application 51
5.1.1 Usability features 51
5.1.2 Routes 52
5.1.3 Efficiency 52
5.1.4 Offline storage and syncing 52
5.1.5 Photos and albums 52
5.1.6 Other media 53
5.1.7 Search 53
5.1.8 Further recommendation system 53
5.2 Navigation 53
5.3 Gamification 54
5.4 Web interface 54
5.5 Administration tools 54
5.6 Social Media 55

6 Results and discussion 57
6.1 Results 57
6.1.1 Research questions 57
6.1.2 Project goals 58
6.2 Discussion 59
6.2.1 Limitations 59
6.2.2 Development methodology 59

7 Conclusion 61

Bibliography 64

A Installation Guide **65**

B Produced Source Code **67**

B.1 Android Client 67

B.2 App Engine 96

B.2.1 Datastore 96

B.2.2 Endpoints 102

Chapter 1

Introduction

This thesis is a continuation of a depth study performed during the spring term 2013 at the Norwegian University of Science and Technology. The thesis started as a very broad topic of utilizing an Android application with crowdsourcing, social media and similar concepts to enable users to learn about art in public spaces. However, the scope of the thesis has been specified significantly since then.

While it is based on a previous project, the relevant information will be summarized in such a way that reading the previous paper will not be necessary.

1.1 Purpose

The purpose of this thesis is two-fold. First, the thesis will conduct research how we can utilize crowdsourcing and geo-location technology in smartphones to build a database of information about art in public places. Secondly, the thesis will describe the design and implementation of an Android application that will enable users to help building and accessing this database. It will look into the potential advantages and drawbacks of using the approach of crowdsourcing to construct this database and propose solutions to efficiently dealing with these potential problems. Of particular interest is the ability to allow users to construct routes, which are an ordered list of artworks with a specific context.

1.2 Motivation

The motivation for this project is to find a way to utilize crowdsourcing and mobile phones to help create attention about art in public places. The problem with public art is that there are seldom any commercial interest in it. On the other hand, galleries, museums and exhibitions usually have ticket sales and marketing campaigns. This means that it is hard to justify spending money on marketing or other ways to create attention about public art.

1.3 Research Questions

While the primary goal of this thesis is to develop an application as a proof of concept, the thesis will also aim to research important questions related to the application's functionality.

1 How can we motivate users to use and contribute to the application?

2 Are there other concepts than the ones mentioned that could be used to improve this application?

3 What can we do to help users find the art they are interested in?

4 What can we learn from other similar applications that can be utilized in this application?

There are 3 methods that will be used to answer these research questions. There will be a literature study. The literature study aims to look into typical user patterns and challenges in crowdsourcing and social media applications. The study will also look into how smartphones can be utilized in learning. Second, there will be a study of other related application that uses one or more of the concepts this project aims to utilize. As more features are implemented in the application, simple user tests will be conducted to receive feedback on the usability of the application. At the same time, users will be asked to fill out short surveys to help prioritize the next feature to be implemented.

1.4 Project goals

The goal of this project is to develop a software artifact. The artifact is an android application that will serve as a proof of concept for a crowdsourcing application. The application is not intended to be a complete product by the end of the project. Thus the functionality to be developed has been narrowed down to the following features:

1 Allow users to add information about artworks to a database

2 Allow users to get information added by other users

3 Allow users to edit or add to information already added to the database

4 Allow users to create routes of several different artworks

5 Assist users in finding artworks or routes they are likely to be interested in

What separates this application from other applications that uses geo-location and crowdsourcing for similar concepts, is the addition of routes. The application also brings together concepts from different applications. It aims to gather knowledge from the crowd and make publicly available, like Wikipedia. But at the same time, it will attempt to utilize location-context and interactivity like the other applications mobile applications we will study.

Naturally, the thesis will also describe why the application is designed as it is and ideas to make the application complete.

This report will attempt to describe the reasoning behind the design of the application's user interface, functionality, the cloud back-end service and the communication between the user client application and the cloud back-end service.

1.5 Development methodology

As there is only a single developer working on the application, the development will take an iterative approach. The plan is to conduct small user tests as new features are implemented, and conduct surveys to prioritize new tasks as tasks are completed.

Chapter 2

Application concept

The intention of this section is to describe the application's functionality and goals in non-technical terms.

2.1 Application's goal

The goal that the application is intended to fulfill is to help create attention around public art, by assisting users in both discovering and learning about public art.

2.2 Functionality

2.2.1 Add, get and edit information

Users should be able to add new artworks and get and edit artworks other users have added. This means that there needs to be a central database that users connect to via internet. Since this is an Android application, users can connect via wi-fi or 3G network.

Users should also be able to rate artworks.

2.2.2 Routes

The addition of routes is critical for this application concept to be a valuable addition to the functionality of geo-location/crowdsourcing applications.

Routes are a simple concept, in general they are mostly an ordered list of artworks. They have their own description and rating.

There are two different ways to create routes. Users can create a route, choose which artworks are included in it and save it to the database. However, users should also be able to be given a route by the application, based on certain criteria. In the scope of this project these criteria should be the top rated artworks, artwork categories and distance.

2.2.3 Assisting users in finding artworks

The application should make it easier for users to discover artworks they might be interested in. For this thesis the implemented features for this will be the ability for users to mark their interest in certain artwork categories, see the top rated artworks, overall and within each category, and the generated routes described in the previous section.

2.3 Requirements

The functionality described earlier can be condensed into the following structured functional requirements.

2.3.1 Functional Requirements

The application shall:

FR1 Allow users to locate artworks on a map

FR2 Allow users to add a new artwork at their current location

FR3 Allow users to describe the artwork by the following items:

- FR3.1 Name
- FR3.2 Artist
- FR3.3 Year of creation
- FR3.4 Category
- FR3.5 A short description
- FR3.6 Historical background
- FR3.7 A photo

FR4 Allow users to see the information about existing artworks:

FR5 Allow users to rate existing artworks

FR6 Allow users to see the average rating of an artwork

FR7 Allow users to add routes consisting of several artworks.

FR8 Describe routes by the following items:

- FR8.1 Name
- FR8.2 A list of artworks
- FR8.3 A short description
- FR8.4 Historical background

FR9 Allow users to rate existing routes

FR10 Allow users to see the average rating of a route

FR11 Allow users to add an artwork to their favorites

FR12 Allow users to see a list of their favorite artworks

2.3.2 Non-Functional Requirements

Modifiability

There's a big chance that this application will either be extended or built upon to keep improving the concept of a geo-location/crowdsourcing application to help create attention to public art. Therefore, modifiability of the application is very important. This means that it should be easy to add or remove specific functionality or features without breaking the application. It also means that the code has to be readable and easy to understand for other developers.

2.3.3 Availability

While availability is extremely important to the user experience, most of the responsibility has been delegated to the cloud service. However, it is still important to provide understandable feedback to the user in case of errors.

Security

The application utilizes functionality built into the Android framework to recognize users. It only uses the email address, and doesn't require a password to log in, so security is not a concern.

Usability

Usability is a high priority for this application. This means both that the application should be intuitive and easy to use for users, but also that when errors happens, the users need to receive understandable error messages and clear instructions on how to fix the error.

While this thesis does not intend to produce a fully functioning complete application, usability should be considered throughout planning and development. To assist in improving usability, some user testing will be conducted during development to receive feedback on the application.

Performance

Performance is not something this project intends to focus on, as the application will not reach the stage of readiness where this is a priority. However, especially network performance will be important in a finished product, as a lot of the user interaction depends on getting and sending data to the server. This means that users have to wait for data for the application to update, and users want to wait as short as time as possible.

Chapter 3

Research

This section will describe the research done before application development starts. It will look at literature and related applications.

3.1 Literature review

The literature review aims to look into typical user patterns in related applications, crowdsourcing platforms and geo-location applications.

3.1.1 Crowdsourcing

Since crowdsourcing is such an integral part of the application, conducting research on the benefits, drawbacks and challenges of crowdsourcing is important.

The term crowdsourcing was first used by Jeff Howe in a now famous article [1] published in Wired Magazine in 2006. Crowdsourcing means outsourcing tasks to a general crowd. Today there are several different examples of major crowdsourcing projects, such as the gigantic online encyclopedia

[1] where both articles and translations are done and moderated by users, or OpenStreetMap [2] which is a world map made by mappers and available to anyone for free. An example mentioned in the original article is the open-source software movement, where the source code of software is available to everyone, either to learn from or to improve upon.

Crowdsourcing provides obvious advantages. It provides access to a very wide variety of knowledge and it's cheap, often free. However, utilizing crowdsourcing also comes with several challenges. A paper about the quality of the data provided to OpenStreetMaps [2] points out several difficulties with validating the information received. One issue is finding a source to compare the data with. Another paper that researches the quality of Wikipedia articles[3] concludes that a simple way to determine the probability of the content having

[1] `www.wikipedia.org`

[2] `http://www.openstreetmap.org/`

a high quality is having more interaction between contributors. This does intuitively make sense - the more people contributing to the content and the more interaction between them, the higher the probability that the content is correct.

Another study on Wikipedia[4] concludes that an important contributor to the quality of articles is the ability for users to discuss the content on a page, and thus being able to challenge and verify information added by other users. It also points out how helpful it is to have a community that cares about the quality of the dataset is. Yet another study [5] shows the importance of so-called "elite" users in the early days of Wikipedia. These users were responsible for a large percent of the content added. But as the dataset contained in Wikipedia grew, so did the contribution from average users. While "elite" users kept providing the same amount of content as before, the percentage of the total amount of content they provided fell dramatically.

Another conclusion made in[4] is that giving extra attention to quality articles through featuring them on the front page inspires users to strive for the same quality when adding their own content.

3.1.2 Attracting and motivating users

A study [6] on Amazon's Mechanical Turk [3] platform shows that a user's perceived "meaning" (i.e. importance) of the tasks they solve increase both the quality, motivation and quantity of the work done on the task. Another study [7] on the same platform concludes that increased payment of completing the task also improves the quality of the work. It also concludes in agreement with [6] that the intrinsic value perceived by the worker increases the quality of the work, at least at low payment values.

On motivating users [8] conducts a study that shows that turning crowdsourcing into a game is an efficient way to motivate users to complete tasks for less, or even no, pay. However, on the topic of gamification [9] shows that while it is an effective strategy to motivate users, care must be taken that it is applied appropriately.

3.1.3 Learning and mobile devices

While not the primary goal for the scope of this thesis, part of the goal of creating this application is to enable learning about the art in public places. A large literature review [10] in mobile technology and learning looks into different ways to incorporate mobile devices in learning. In Figure 3.1 [4] we see that a large percentage of the population in Norway has access to mobile phones, and over half are smartphone users. The most likely way the application being developed in this thesis could be applied according to the activities described in [10] would be "supporting intentional and accidental learning episodes" by providing users information while on the move. In a study[11] 48 students were

[3]`https://www.mturk.com/mturk/`

[4]`https://www.ssb.no/statistikkbanken/SelectVarVal/Define.asp?MainTable=MedieElektron&KortNavnWeb=medie&PLanguage=1&checked=true`

given PDAs with information and pictures while visiting a museum. The PDAs allowed the students to follow trails through the museum. The study concluded that the students experienced positive results in learning, with the technical aspect of using PDAs being the main limiting factor. With the availability of mobile devices and smartphones as shown in Figure 3.1, students could use their phone as a familiar device while following a trail like the ones described in the[11] study.

A study[12] conducted on the use of mobile devices for learning points out the advantaged of using familiar devices, as this naturally improves the usability of the devices. We could imagine the application proposed in this thesis being used by a school class to provide an interactive learning experience through their home city.

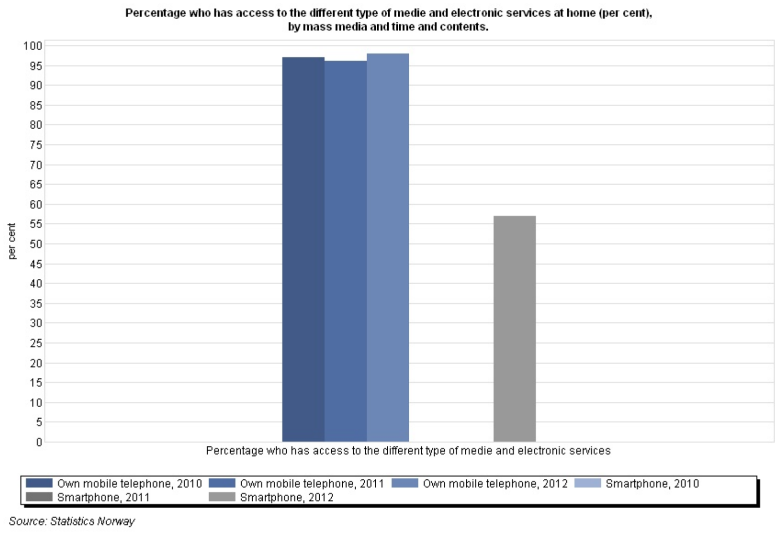


Figure 3.1: Access to mobile phones and smartphones in Norway 2010-2012.

3.2 Related applications

This section describes the related applications that have been researched and what relevance they have for the application being developed in the thesis.

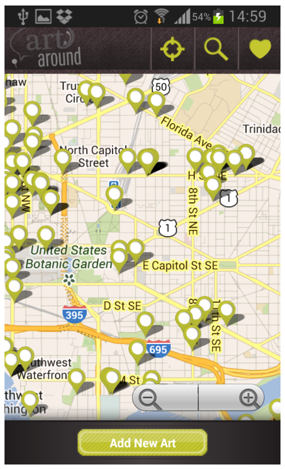
3.2.1 ArtAround

Unfortunately, it seems that this application has been removed from the Google Play Store during the project, and is now only available on the iPhone. However, the website is still accessible, and screenshots were taken before it was removed, so the research conducted will still be applied to the thesis. The application is open-source.

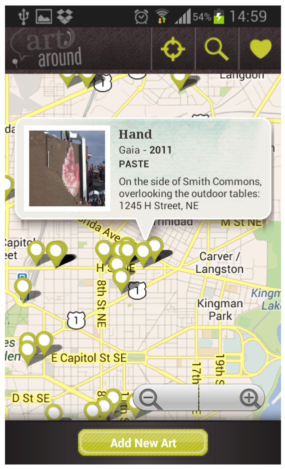
Website: `http://theartaround.us/`
GitHub repository: `https://github.com/ArtAround`

Description

ArtAround is the most similar application to the concept of this thesis, in both the goal of the application and the functionality. It utilities both crowdsourcing and geo-location.



(a) Map screen in ArtAround



Hand
Gaia - 2011
PASTE
On the side of Smith Commons, overlooking the outdoor tables: 1245 H Street, NE

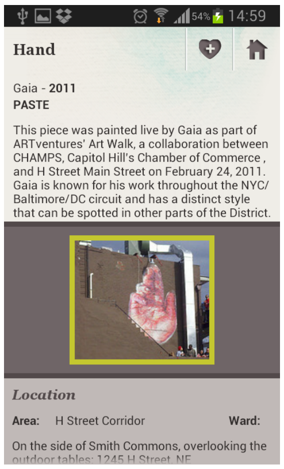
(b) Information fragment on touch

Figure 3.2: The map screen in ArtAround

Like our concept, ArtAround uses a map where each marker is an artwork. Pressing a marker brings up an information fragment on top of the map that shows a picture, the address and a short description of the artwork as shown in Figure3.2. Pressing this information fragment brings the user to a new page where they can view more detailed information about the artwork, view several pictures if they exist and read other user's comment about the artwork as shown in Figure 3.3

Pressing the add new art button brings the user to the screen in Figure 3.4). In this screen the user can input the art's name, artist, year, category, a description, as well as an area and a ward and a description of the location. The user can also add a photo of the art.

ArtAround allows users to comment on existing art, which are shown at the bottom of the art's information page as shown in Figure 3.5 and reading comments as shown in Figure 3.6


54% 14:59

Hand

Gaia - **2011**
PASTE

This piece was painted live by Gaia as part of ARTventures' Art Walk, a collaboration between CHAMPS, Capitol Hill's Chamber of Commerce , and H Street Main Street on February 24, 2011. Gaia is known for his work throughout the NYC/ Baltimore/DC circuit and has a distinct style that can be spotted in other parts of the District.

Location

Area: H Street Corridor **Ward:**

On the side of Smith Commons, overlooking the outdoor tables: 1245 H Street, NE

Figure 3.3: Art information screen in ArtAround

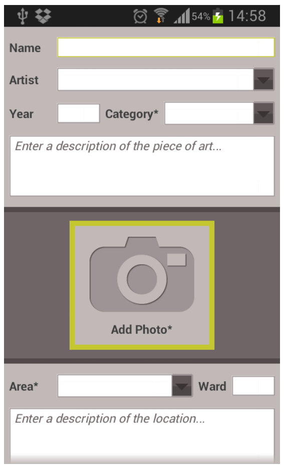
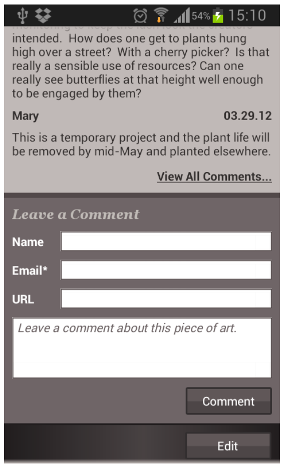

54% 14:58
Name
Artist
Year
Category*
Enter a description of the piece of art...
Add Photo*
Area*
Ward
Enter a description of the location...

Figure 3.4: Adding new art in ArtAround

The application includes the option for users to favorite art. On the art information page a user can press the favorite button, represented by a heart icon, to add the art to their favorites. The user can view all their favorites from the map screen. This filters away all art on the map screen that is not part of the favorite list, and the resulting screen looks like what is shown in Figure 3.7

Relevance

Among the researched applications, ArtAround is the application that is most similar to the concept described in this thesis. Letting users add, edit and find art functions in a similar manner envisioned in this application. However, it



15:10

intended. How does one get to plants hung high over a street? With a cherry picker? Is that really a sensible use of resources? Can one really see butterflies at that height well enough to be engaged by them?

Mary **03.29.12**

This is a temporary project and the plant life will be removed by mid-May and planted elsewhere.

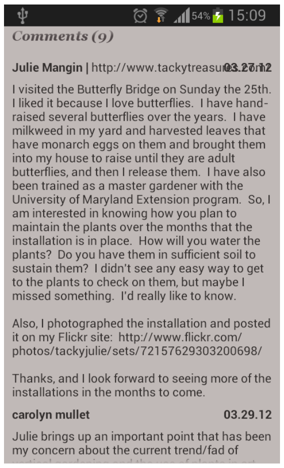
View All Comments...

Leave a Comment

Name	
Email*	
URL	

Leave a comment about this piece of art.

Figure 3.5: Adding a comment in ArtAround



15:09

Comments (9)

Julie Mangin | http://www.tackytreasu**03.27.12**

I visited the Butterfly Bridge on Sunday the 25th. I liked it because I love butterflies. I have hand-raised several butterflies over the years. I have milkweed in my yard and harvested leaves that have monarch eggs on them and brought them into my house to raise until they are adult butterflies, and then I release them. I have also been trained as a master gardener with the University of Maryland Extension program. So, I am interested in knowing how you plan to maintain the plants over the months that the installation is in place. How will you water the plants? Do you have them in sufficient soil to sustain them? I didn't see any easy way to get to the plants to check on them, but maybe I missed something. I'd really like to know.

Also, I photographed the installation and posted it on my Flickr site: http://www.flickr.com/photos/tackyjulie/sets/72157629303200698/

Thanks, and I look forward to seeing more of the installations in the months to come.

carolyn mullet **03.29.12**

Julie brings up an important point that has been my concern about the current trend/fad of

Figure 3.6: Reading comments in ArtAround

does not include the concept of routes, top rated artworks or the ability for users to choose their interests by category. It is also quite obviously geared towards american cities, as areas are pre-defined, and are connected to a ward.

3.2.2 FourSquare

FourSquare is an application that aims to help people find ”places”. In foursquare’s context ”places” includes places you can visit, like restaurants, cafes, nightclubs, parks, scenic places, shopping, entertainment, etc. Figure 3.8 shows most of the types of places a user can select to search for in the application.

When users first start the application, they are required to create an account.

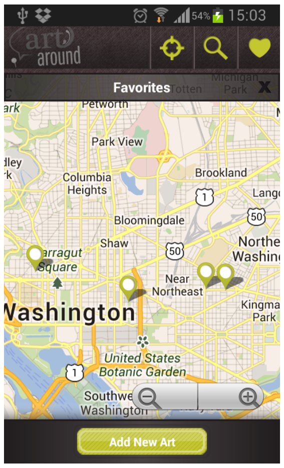
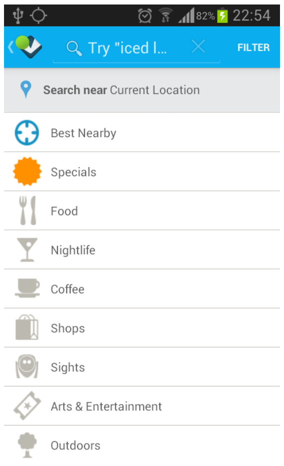


Figure 3.7: Map with favorites enabled



Search near Current Location
Best Nearby
Specials
Food
Nightlife
Coffee
Shops
Sights
Arts & Entertainment
Outdoors

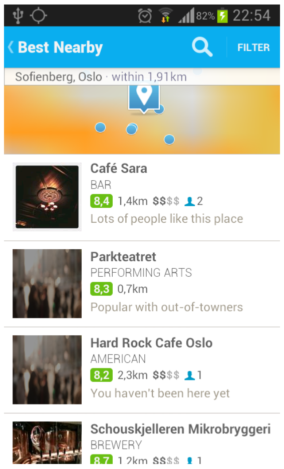
Figure 3.8: The categories of places in foursquare

However, they can sign in directly with facebook or twitter accounts.

Foursquare relies on users searching for places, instead of just browsing the map. The search allows you choose a category, like food, nightlife, coffee, shopping, etc. It also allows you to choose "best nearby", which shows you the highest rated nearby places of any category as shown in Figure 3.9.

The search can also filter on more things, like distance, show only visited/unvisited places, etc. In Figure 3.10 we see some of the more advanced filtering options provided.

When a search is completed, the map is updated with blue circles showing the places that are returned and the news feed is replaced by a list of the places



Sofienberg, Oslo · within 1,91km

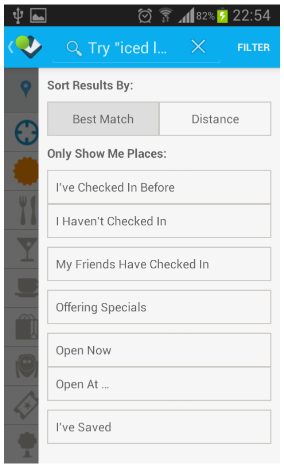
Café Sara
BAR
8,4 1,4km $$$$ 2
Lots of people like this place

Parkteatret
PERFORMING ARTS
8,3 0,7km
Popular with out-of-towners

Hard Rock Cafe Oslo
AMERICAN
8,2 2,3km $$$$ 1
You haven't been here yet

Schouskjelleren Mikrobryggeri
BREWERY

Figure 3.9: The best nearby list in



Sort Results By:
Best Match | Distance

Only Show Me Places:
I've Checked In Before
I Haven't Checked In
My Friends Have Checked In
Offering Specials
Open Now
Open At ...
I've Saved

Figure 3.10: Additional search filters in foursquare

returned by the search. Pressing the map will make the list disappear and the map fills the screen as shown in Figure3.11, while pressing an item in the list will make the map zoom to that place. Pressing the place again will bring the user to a new screen, where more information about this place is shown.

The place's information screen shows of many of foursquare's features:

Save Foursquare allows users to save places to their to-do list, which is meant to be a list of places they want to visit in the future.

Check-in Users can check-in to a place. This allows you to share with other users what you did her, add a photo, tag a friend and share the check-in on facebook and/or twitter.

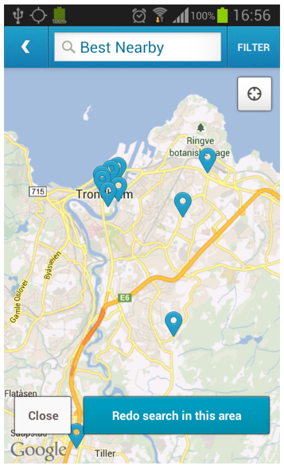


Figure 3.11: Map showing search results in foursquare

Leave a tip Leave a tip enables users to leave a short comment about the place. This could be a particularly good dish at a restaurant, etc.

Like/Dislike Users can like/dislike places to improve the recommendations foursquare gives them.

View lists Foursquare allows users to create custom lists. From the place's information screen you can get a list of all the lists this place are in.

Mayor "Mayor" is a part of foursquare's gamification elements, and is awarded to the user that has checked into a place the most times during the last 60 days.

Suggest an edit The user can suggest changes or additions to the place's information, but this has to be approved.

We can see part of the place screen in Figure 3.12 while the list interface can be seen in Figure 3.13

A recent update also added a gamification element to foursquare. Users can earn badges by using the application. The badges are given for completing tasks like visiting a place, rating, editing information, etc. The badge screen is shown in Figure 3.14

Lastly, foursquare has it's own social network. Users can import friends from other social networks like twitter and facebook, but will provide a special newsfeed for friends' activity in foursquare. Figure 3.15 shows how you can find friends in foursquare.

Relevance Foursquare is the application that has had features that are most similar and relevant to the concept application in this thesis.

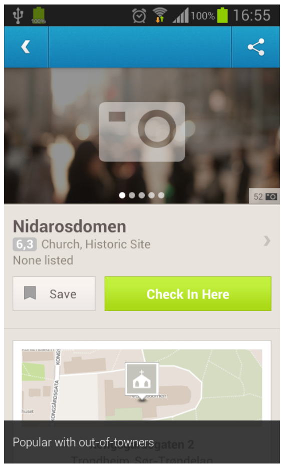
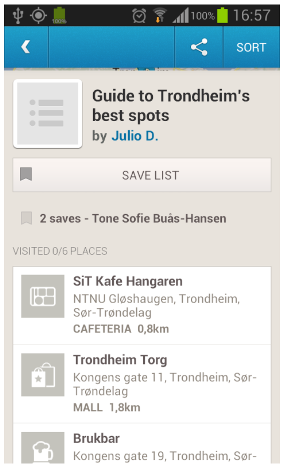


Figure 3.12: A place in foursquare


16:57

SORT

Guide to Trondheim's best spots
by Julio D.

SAVE LIST

2 saves - Tone Sofie Buås-Hansen

VISITED 0/6 PLACES

SiT Kafe Hangaren
NTNU Gløshaugen, Trondheim, Sør-Trøndelag
CAFETERIA 0,8km

Trondheim Torg
Kongens gate 11, Trondheim, Sør-Trøndelag
MALL 1,8km

Brukbar
Kongens gate 19, Trondheim, Sør-

Figure 3.13: A list in foursquare

The use of lists is of particular interest to this thesis. The ability to create custom lists and see which lists a venue is a part of is similar to the idea of routes in the application. However, this application goes a step further, enabling routes to contain a strict order of artworks, or suggest the best order to visit each artwork in a route in.

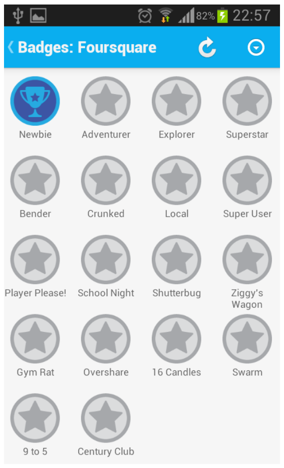


Figure 3.14: Badges in foursquare

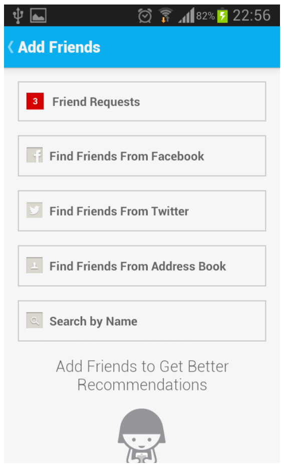

22:56
Add Friends
3
Friend Requests
Find Friends From Facebook
Find Friends From Twitter
Find Friends From Address Book
Search by Name
Add Friends to Get Better Recommendations

Figure 3.15: Social network in foursquare

3.2.3 Waze

Waze is navigation application for both iOS and Android. It provides turn-by-turn navigation while driving, but also allows users to share other data, like gas prices, traffic congestions, road work, etc. The add report screen is shown in Figure 3.16, and the report is automatically added to the current location of the user.

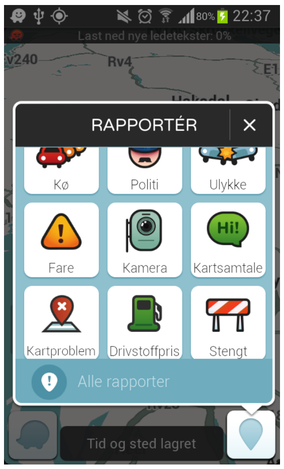


Figure 3.16: Adding a report in Waze

Waze is a little different from the other applications reviewed, as it doesn't utilize things like "places" or similar. However, it does utilize a concept referred to as "passive crowdsourcing". This means that users don't have to actively provide information. If users choose to do so, they can passively provide information to Waze while their phone is connected to the internet. The application will then gather the user's average speed, and provide this information to other users. This means that when using the app for navigation, you can also see the speed with which other users are moving, as shown in Figure3.17. This way users can anticipate traffic congestion and decide to take another road if they wish to.

Like many other crowdsourcing applications, Waze also offers several gamification elements. Users receive points for completing tasks, like adding new content, adding friends, etc. Receiving points allows users to level up. Leveling up again gives users access to new avatars and trophies, so they can show off to other users. Waze's user profile score is shown in Figure 3.18.

One particularly interesting feature in Waze is that the application will generate tasks for users to visit certain areas, as shown in Figure 3.19. This motivates users to visit areas where the application is missing data, or haven't received updated data from in a long time.

Waze also offers turn-by-turn navigation as shown in Figure 3.20. This is unique for the applications studied for this thesis.

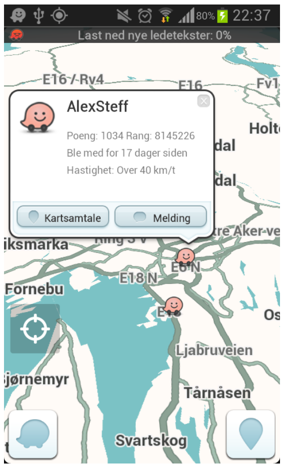
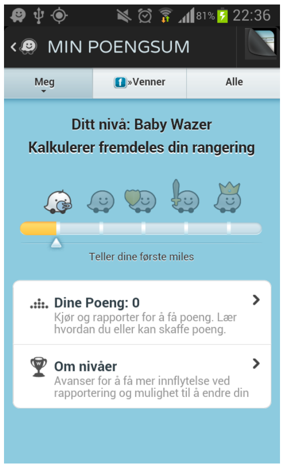


Figure 3.17: A user with their score and avatar in Waze

Relevance While foursquare's user interface and structure of places is more relevant to the application being developed in this thesis, Waze does have some interesting features. Especially the ability to provide turn by turn navigation might be useful to help users discover art. Waze also has an interesting approach in using gamification to motivate users to add information that is needed in the application.



Ditt nivå: Baby Wazer
Kalkulerer fremdeles din rangering

Teller dine første miles

Dine Poeng: 0
Kjør og rapporter for å få poeng. Lær hvordan du eller kan skaffe poeng.

Om nivåer
Avanser for å få mer innflytelse ved rapportering og mulighet til å endre din

Figure 3.18: The user profile showing the score you have achieved

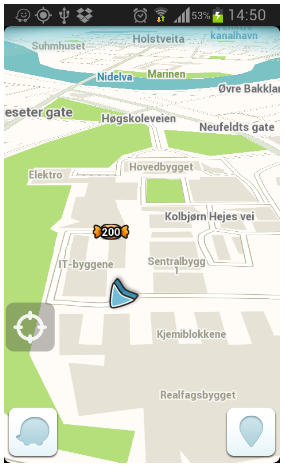


Figure 3.19: A task to visit a place in Waze

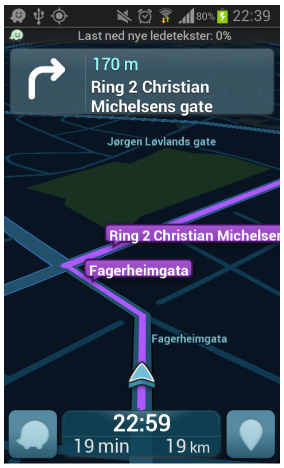
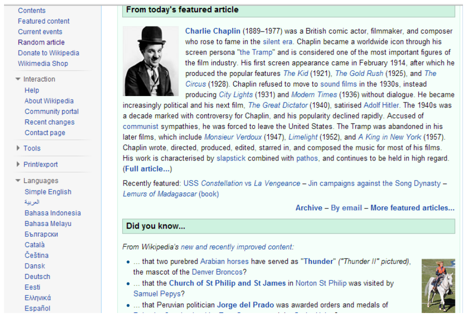

22:39
Last ned nye ledetekster: 0%
170 m
Ring 2 Christian Michelsens gate
Jørgen Løvlands gate
Ring 2 Christian Michelsen
Fagerheimgata
Fagerheimgata
22:59
19 min
19 km

Figure 3.20: The turn by turn interface in Waze

3.2.4 Wikipedia

Although not a mobile application, the application to be developed in this thesis shares some features with Wikipedia when it comes to building a database based on crowdsourcing. The literature study showed that several studies have been done on the quality of data added to Wikipedia.



From today's featured article

Charlie Chaplin (1889–1977) was a British comic actor, filmmaker, and composer who rose to fame in the silent era. Chaplin became a worldwide icon through his screen persona "the Tramp" and is considered one of the most important figures of the film industry. His first screen appearance came in February 1914, after which he produced the popular features *The Kid* (1921), *The Gold Rush* (1925), and *The Circus* (1928). Chaplin refused to move to sound films in the 1930s, instead producing *City Lights* (1931) and *Modern Times* (1936) without dialogue. He became increasingly political and his next film, *The Great Dictator* (1940), satirised Adolf Hitler. The 1940s was a decade marked with controversy for Chaplin, and his popularity declined rapidly. Accused of communist sympathies, he was forced to leave the United States. The Tramp was abandoned in his later films, which include *Monsieur Verdoux* (1947), *Limelight* (1952), and *A King in New York* (1957). Chaplin wrote, directed, produced, edited, starred in, and composed the music for most of his films. His work is characterised by slapstick combined with pathos, and continues to be held in high regard. (**Full article...**)

Recently featured: USS *Constellation* vs *La Vengeance* – Jin campaigns against the Song Dynasty – *Lemurs of Madagascar* (book)

Archive – **By email** – **More featured articles...**

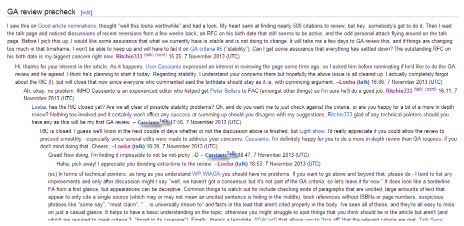
Did you know...

From Wikipedia's new and recently improved content:

- ... that two purebred Arabian horses have served as "**Thunder**" *("Thunder II" pictured)*, the mascot of the Denver Broncos?
- ... that the **Church of St Philip and St James** in Norton St Philip was visited by Samuel Pepys?
- ... that Peruvian politician **Jorge del Prado** was awarded orders and medals of

Figure 3.21: Wikipedia's front page

Figure 3.21 shows the front page of Wikipedia. Here we can see some of the features discussed in the literature review, particularly the use of the featured article. The featured article is highlighted to give users an example of the quality of content the Wikipedia staff wishes for.



GA review precheck [edit]

I saw this on Good article nominations, thought "well this looks worthwhile" and had a look. My heart sank at finding nearly 500 citations to review, but hey, somebody's got to do it. Then I read the talk page and noticed discussions of recent reversions from a few weeks back, an RFC on his birth date that still seems to be active, and the odd personal attack flying around on the talk page. Before I pick this up, I would like some assurance that what we currently have is *stable* and is not doing to change. It will take me a few days to GA review this, and if things are changing too much in that timeframe, I won't be able to keep up and will have to fail it on GA criteria #5 ("stability"). Can I get some assurance that everything has settled down? The outstanding RFC on his birth date is my biggest concern right now. **Ritchie333** (talk) (cont) 15:25, 7 November 2013 (UTC)

Hi, thanks for your interest in the article. As it happens, User:Cassianto expressed an interest in reviewing the page some time ago, so I asked him before nominating if he'd like to do the GA review and he agreed. I think he's planning to start it today. Regarding stability, I understand your concerns there but hopefully the above issue is all cleared up. I actually completely forgot about the RfC (!), but will close that now since everyone who commented said the birthdate should stay as it is, with convincing argument. --**Loeba** **(talk)** 16:06, 7 November 2013 (UTC)

Ah, okay, no problem. IMHO Cassianto is an experienced editor who helped get Peter Sellers to FAC (amongst other things) so I'm sure he'll do a good job. **Ritchie333** (talk) (cont) 16:11, 7 November 2013 (UTC)

Loeba, has the RfC closed yet? Are we all clear of possible stability problems? Oh, and do you want me to *just* check against the criteria, or are you happy for a bit of a more in depth review? Nothing too involved and it certainly won't effect any success at summing up should you disagree with my suggestions. Ritchie333 glad of any technical pointers should you have any as this will be my first GA review. -- **Cassianto**Talk 17:56, 7 November 2013 (UTC)

RfC is closed. I guess we'll know in the next couple of days whether or not the discussion above is finished, but Light show, I'd really appreciate if you could allow the review to proceed smoothly - especially since several edits were made to address your concerns. Cassianto, I'm definitely happy for you to do a more in-depth review than GA requires, if you don't mind doing that. Cheers, --**Loeba** **(talk)** 18:39, 7 November 2013 (UTC)

Great! Now doing, I'm finding it impossible to not be not-picky :-D -- **Cassianto**Talk 18:47, 7 November 2013 (UTC)

Haha, pick away! I appreciate you devoting extra time to the review. --**Loeba** **(talk)** 18:53, 7 November 2013 (UTC)

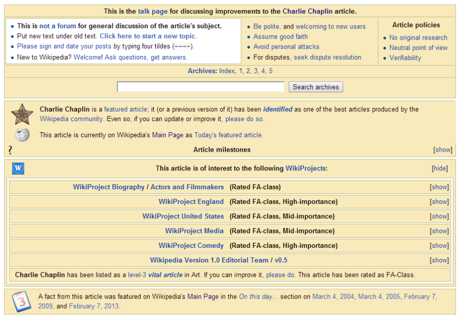
(ec) In terms of technical pointers, as long as you understand WP:WIAGA you should have no problems. If you want to go above and beyond that, please do - I tend to list *any* improvements and only after discussion might I say "well, we haven't got a consensus but it's not part of the GA criteria, so let's leave it for now." It does look like a borderline FA from a first glance, but appearances can be deceptive. Common things to watch out for include checking ends of paragraphs that are uncited, large amounts of text that appear to only cite a single source (which may or may not mean an uncited sentence is hiding in the middle), book references without ISBNs or page numbers, suspicious phrases like "some say", "most claim", "... is universally known to" and facts in the lead that aren't cited properly in the body. I've seen all of these, and they're all easy to miss on just a casual glance. It helps to have a basic understanding on the topic, otherwise you might struggle to spot things that you think should be in the article but aren't (and which are required to meet criteria 3, "broad in its coverage"). Finally, there's a template, {{GAList}} that allows you to "tick off" that the relevant criteria are met. Talk:Sega

Figure 3.22: Wikipedia talk page

Figure 3.22 shows what the "talk" page of Wikipedia looks like. Here we can see users discussing changes to a page, either content that they mean should be added or removed, or users volunteering to review the sources and citations of the page.

Figure 3.23 shows another part of Wikipedia's talk page of the same article. Here we see can see how Wikipedia grades the article as a high quality article, but still encourages users to add content if they feel they can improve upon it.

In Figure 3.24 the revision history of another featured article is shown. This allows users to track changes to the article. This way, every user can review changes made, and start a discussion on the changes made if they disagree.



This is the talk page for discussing improvements to the Charlie Chaplin article.

- **This is not a forum for general discussion of the article's subject.**
- Put new text under old text. **Click here to start a new topic.**
- Please sign and date your posts by typing four tildes (~~~~).
- New to Wikipedia? Welcome! Ask questions, get answers.

- Be polite, and welcoming to new users
- Assume good faith
- Avoid personal attacks
- For disputes, seek dispute resolution

Article policies

- No original research
- Neutral point of view
- Verifiability

Archives: Index, 1, 2, 3, 4, 5

Search archives

Charlie Chaplin is a featured article; it (or a previous version of it) has been ***identified*** as one of the best articles produced by the Wikipedia community. Even so, if you can update or improve it, please do so.

This article is currently on Wikipedia's Main Page as Today's featured article.

Article milestones [show]

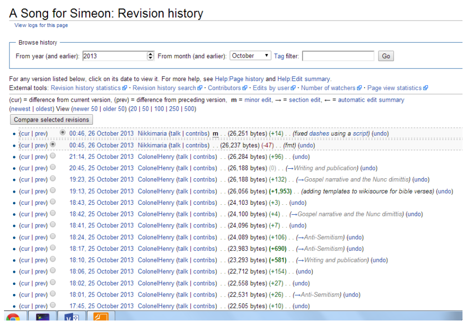
This article is of interest to the following WikiProjects: [hide]

WikiProject Biography / Actors and Filmmakers	**(Rated FA-class)**	[show]
WikiProject England	**(Rated FA-class, High-importance)**	[show]
WikiProject United States	**(Rated FA-class, Mid-importance)**	[show]
WikiProject Media	**(Rated FA-class, Mid-importance)**	[show]
WikiProject Comedy	**(Rated FA-class, High-importance)**	[show]
Wikipedia Version 1.0 Editorial Team / v0.5		[show]

Charlie Chaplin has been listed as a level-3 ***vital article*** in Art. If you can improve it, please do. This article has been rated as FA-Class.

A fact from this article was featured on Wikipedia's Main Page in the *On this day...* section on March 4, 2004, March 4, 2005, February 7, 2009, and February 7, 2013.

Figure 3.23: Wikipedia talk page grading



A Song for Simeon: Revision history

View logs for this page

Browse history

From year (and earlier): 2013 From month (and earlier): October Tag filter: Go

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary.
External tools: Revision history statistics · Revision history search · Contributors · Edits by user · Number of watchers · Page view statistics

(cur) = difference from current version, (prev) = difference from preceding version, **m** = minor edit, → = section edit, ← = automatic edit summary
(newest | oldest) View (newer 50 | older 50) (20 | 50 | 100 | 250 | 500)

Compare selected revisions

- (cur | prev) 00:46, 26 October 2013 Nikkimaria (talk | contribs) **m** . . (26,251 bytes) (+14) . . *(fixed dashes using a script)* (undo)
- (cur | prev) 00:45, 26 October 2013 Nikkimaria (talk | contribs) . . (26,237 bytes) (-47) . . *(fmt)* (undo)
- (cur | prev) 21:14, 25 October 2013 ColonelHenry (talk | contribs) . . (26,284 bytes) (+96) . . (undo)
- (cur | prev) 20:45, 25 October 2013 ColonelHenry (talk | contribs) . . (26,188 bytes) (0) . . *(→Writing and publication)* (undo)
- (cur | prev) 19:23, 25 October 2013 ColonelHenry (talk | contribs) . . (26,188 bytes) (+132) . . *(→Gospel narrative and the Nunc dimittis)* (undo)
- (cur | prev) 19:13, 25 October 2013 ColonelHenry (talk | contribs) . . (26,056 bytes) **(+1,953)** . . *(adding templates to wikisource for bible verses)* (undo)
- (cur | prev) 18:43, 25 October 2013 ColonelHenry (talk | contribs) . . (24,103 bytes) (+3) . . (undo)
- (cur | prev) 18:42, 25 October 2013 ColonelHenry (talk | contribs) . . (24,100 bytes) (+4) . . *(→Gospel narrative and the Nunc dimittis)* (undo)
- (cur | prev) 18:41, 25 October 2013 ColonelHenry (talk | contribs) . . (24,096 bytes) (+7) . . (undo)
- (cur | prev) 18:24, 25 October 2013 ColonelHenry (talk | contribs) . . (24,089 bytes) (+106) . . *(→Anti-Semitism)* (undo)
- (cur | prev) 18:17, 25 October 2013 ColonelHenry (talk | contribs) . . (23,983 bytes) **(+690)** . . *(→Anti-Semitism)* (undo)
- (cur | prev) 18:10, 25 October 2013 ColonelHenry (talk | contribs) . . (23,293 bytes) **(+581)** . . *(→Writing and publication)* (undo)
- (cur | prev) 18:06, 25 October 2013 ColonelHenry (talk | contribs) . . (22,712 bytes) (+154) . . (undo)
- (cur | prev) 18:02, 25 October 2013 ColonelHenry (talk | contribs) . . (22,558 bytes) (+27) . . (undo)
- (cur | prev) 18:01, 25 October 2013 ColonelHenry (talk | contribs) . . (22,531 bytes) (+26) . . *(→Anti-Semitism)* (undo)
- (cur | prev) 17:45, 25 October 2013 ColonelHenry (talk | contribs) . . (22,505 bytes) (+10) . . (undo)

Figure 3.24: Wikipedia article revision history

Relevance While not as technically relevant as the mobile applications reviewed in this section, there are several lessons to be learned from Wikipedia. Combining the literature reviewed and the examples shown here, we can see the importance of attracting elite users that are interested in creating quality content. The content added by these users can then be highlighted to set the bar for new content to be added.

As we saw in the literature study, Wikipedia is probably one of the best crowdsourcing examples to study, if one is interested in creating quality content for without paying participants. Promoting discussion, proper research and source revision are important to constantly improve the quality of the content.

3.3 Previously implemented features

As mentioned in the introduction, this thesis is based on a project conducted during the spring term in 2013. This subsection will summarize the features that were implemented during that project.

The previous project intended to implement most of the basic functionality needed to be able to continue with the route and user account implementation for this thesis.

3.3.1 Data storage

The database implemented in the previous project was a Google App Engine application. This is an application that runs on Google infrastructure and automatically scales as needed. Google App Engine applications are very easy to integrate with Android applications, which was the main reason this was chosen as a back-end.

The only entity that was implemented was the ArtWork entity. This entity consisted of the following fields:

Name	The name of the artwork
Artist	The name of the artist that made the artwork
Year of creation	The year the artwork was made
Category	A set of pre-defined categories the artwork can belong to.
Short description	A short description of the artwork that is shown on the map screen information fragment
Long description	A longer description that is shown on the artwork information page.

The datastore was implemented using JDO annotations. This has now been changed to another annotation library called Objectify. This was done because it was very hard to manipulate data storage in JDO, and objectify provides a lower level interaction with the datastore.

3.3.2 Network

The project used a library to generate a HTTP API to handle sending and getting data from the datastore over an internet connection. The same library is used in this project, but has been extended with more methods, and will be described further in the implementation details section.

3.3.3 Android application

Most of the development effort in the previous project was spent on creating the user interface in the android application. The implemented features were:

1. Actionbar to navigate application
2. The map screen.
   (a) Get all artworks and create a marker for them
   (b) On marker click, show small information fragment
   (c) Clicking on information fragment brings user to artwork information screen
3. Add artwork screen
   (a) Automatically gets longitude and latitude
   (b) All the required textfields
   (c) A dropdown box to select category
4. Artwork information screen

Chapter 4

Implementation details

This section will describe the planning and implementation in technical detail. It will show off the implemented features with screenshots, description, explain the way they work and the interaction between them.. For further detail, the source code is included in Appendix B.

4.1 Architecture description

The project technically consists of two applications.

1. The back-end:
   (a) The datastore
   (b) The HTTP API that allows the datastore to be accessed over a network connection
2. The Android client

Figure 4.1 illustrates the architecture and interaction between these applications.

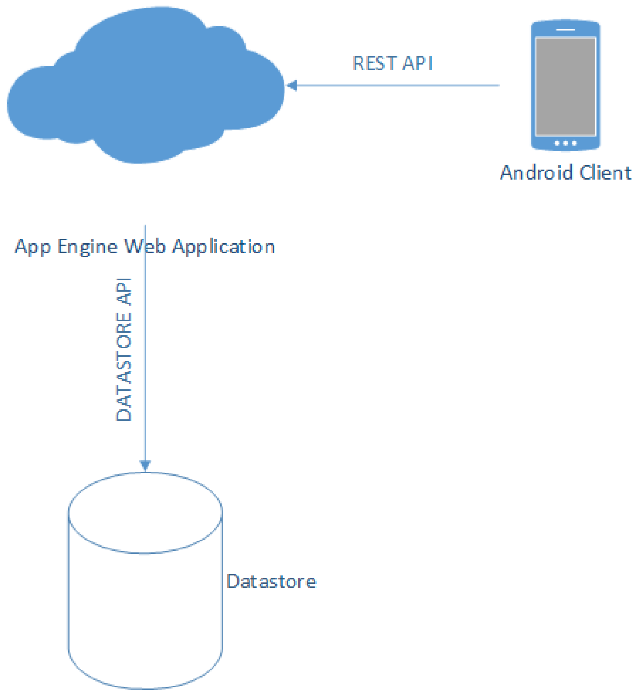


Figure 4.1: Architecture

4.1.1 Datastore

The datastore contains the following entities:

1. ArtWork
2. UserAccount
3. ArtworkRoute

ArtWork

The ArtWork entity contains the following fields:

Id This field is the primary key for each artwork. It is therefore unique for each entry, and auto-assigned by the datastore when a new artwork is created. **Type: Long**

Name The name of the artwork. **Type: String**

Artist The name of the artist that created the artwork. **Type: String**

Category The category of artwork. In the client, this is selected from a predefined list. **Type: String**

Year of creation The year the artwork was created. **Type: Integer**

Short Description A short summary of the artwork to be shown in the information fragment. **Type: Text**

History A longer description of the artwork, intending to explain the history and background of the artwork. **Type: Text**

Latitude The GPS-coordinate latitude of the artwork. **Type: Float**

Longitude The GPS-coordinate longitude of the artwork. **Type: Float**

Ratings This field is slightly more complicated. It contains a list of an embedded class. The embedded class consists of the fields:

> **UserEmail** The email of the user that added the rating. **Type: String**
>
> **Rating** The rating the user gave the artwork. **Type: Integer**
>
> The way this list ends up being stored in the datastore is as two lists with a strict order. This way we ensure that the first entry in the list of user emails corresponds to the first entry in the user ratings, and so forth.

Average rating This field is updated each time a new rating entry is added or updated in the list of Ratings. It calculates the average rating of the artwork. **Type: Integer**

UserAccount

The UserAccount entity's function is to be able to remember users, so we can keep track of their ratings. It contains the following fields:

UserEmail The primary key that uniquely identifies a user. It is an email address selected by the user in the client application. **Type: String**

Favorites A list containing the IDs of ArtWorks the user has added to their favorites. **Type: List<Long>**

ArtworkRoute

The datastore allows users to save routes of artwork they create manually. The entity contains the following fields:

Id This field is the primary key for each route. It is therefore unique for each entry, and auto-assigned by the datastore when a new route is created. **Type: Long**

Name The name of the route. **Type: String**

Artworks A list of the ids of the artworks in the route. **Type: List<Long>**

Ratings This field is slightly more complicated. It contains a list of an embedded class. The embedded class consists of the fields:

> **UserEmail** The email of the user that added the rating. **Type: String**
>
> **Rating** The rating the user gave the route. **Type: Integer**
>
> The way this list ends up being stored in the datastore is as two lists with a strict order. This way we ensure that the first entry in the list of user emails corresponds to the first entry in the user ratings, and so forth.

Average rating This field is updated each time a new rating entry is added or updated in the list of Ratings. It calculates the average rating of the route. **Type: Integer**

Short Description A short summary of the route. **Type: Text**

History A longer description of the route, intending to explain the history and background of the route. **Type: Text**

This datastore model is illustrated in Figure 4.2.

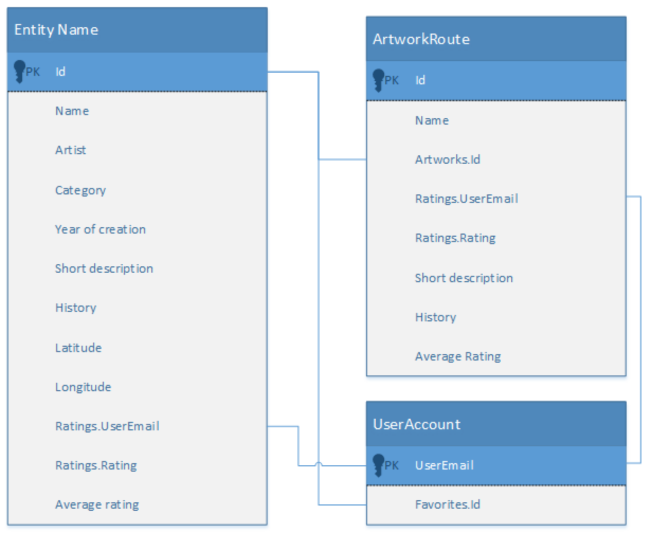


Figure 4.2: Datastore model

4.1.2 Data transfer

To transfer data between the client and the datastore, a library called Google Cloud Endpoints[1] was used. This tool is integrated with Google App Engine, and is therefore somewhat simple to start using. By annotating our code with JDO[2] or Objectify[3] annotations, the tool can auto-generate methods in the back-end and client to:

1. List all entities of a kind
2. Get an entity by it's id
3. Delete an entity by it's id
4. Edit an entity by it's id
5. Add a new entity

In addition, methods to insert user ratings, get top rated items, get user ratings, get an artwork's name by passing id, add favorites and get favorites has been added.

The methods to do these operations will typically have a return type (or void for inserting/updating/deleting), a specified HTTP method and required parameters. The method will then conduct the database call by querying and/or manipulating the datastore, before returning the result if not a void method.

As an illustration, here is the code for the addFavorite endpoint method:

[1] `https://developers.google.com/appengine/docs/java/endpoints/`
[2] `https://developers.google.com/appengine/docs/java/datastore/jdo/overview-dn2`
[3] `https://code.google.com/p/objectify-appengine/`

```
/**
 * This method adds an artwork id to the list of favorites
 * It uses HTTP PUT
 *
 * @param id the id of the artwork to be added
 * @param useraccount the entity to add the artwork id to
 */

@ApiMethod(name = "addFavorite", httpMethod = "PUT", path = "addFavorite"
     )
public void addFavorite(@Named("id") Long id, @Named("accountName")
     String accountName) {
        UserAccount userAccount = ofy().load().type(UserAccount.class).id(
             accountName).get();
        userAccount.addFavorite(id);
        ofy().save().entity(userAccount).now();
}
```

To see all the endpoints method, see Appendix B.

4.1.3 Android client

The Android client is the application that users will interact with. This application targets Android version 3.0 and newer. This is mainly because version 3.0 implemented the actionbar, and creating an actionbar for older versions of the operating system requires more external libraries.

User Interface

The user interface is divided into the different screens that users will interact with. There is also an actionbar that is consistent across every screen.

The interaction between these screens are shown in figure 4.3

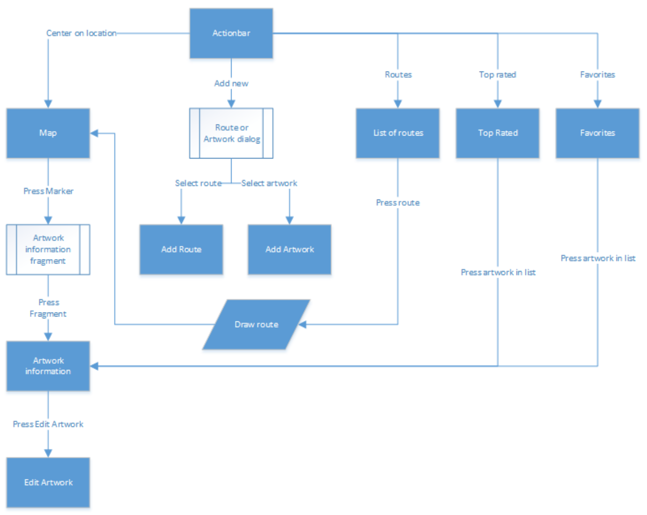


Figure 4.3: The interaction between interface elements

The actionbar The actionbar is the primary means for users to navigate between most of the screens in the application. It is consistent across all screens, to make it easy for users to use.

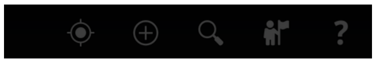


Figure 4.4: The actionbar

Figure 4.4 shows how the actionbar looks in the implemented application. Figure 4.5 shows a closer explanation of the buttons in the actionbar.

The buttons have the following functionality:

Map/center on location This button will bring the user to the map screen. If they are already on the map screen it will center on their current location.

Add Artwork/Route When the user presses this button they will be shown the dialog depicted in Figure 4.13 which queries the user whether they wish to add a route or an artwork. After making their choice and clicking the ok-button, the application goes to the respective creation screen.

List Routes This button takes the user to the screen containing the list of all routes.

List Favorites This button takes the user to the screen containing the list of all their favorite artworks.

Help This button takes the user to the help screen.

If more buttons were to be added to the actionbar, they would be shown in an overflow menu as shown in 4.6.

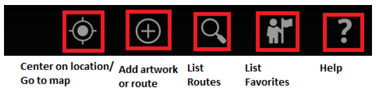


Figure 4.5: The actionbar buttons explained

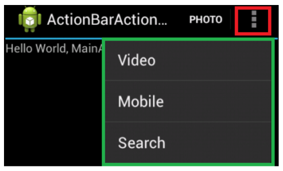


Figure 4.6: Illustration of the overflow functionality

The map screen The map screen is the start screen of the application. The map screen is the primary way for users to look for a new artwork as long as search is not implemented. Currently the application loads all the available artworks in the datastore and marks each one with a marker on the map. As described in the application details section, pressing the marker shows a small information fragment about the artwork.

The first time the user loads the application, they are prompted to choose an account to identify with, as shown in Figure 4.7. In the current implementation, the application only allows users to choose a Google account, but lists all the Google accounts the user has registered on the device.

It is possible to extend this method to allow other types of accounts too, such as social media account like Facebook or Twitter. This functionality utilizes the AccountPicker[4] provided by the Android Framework.

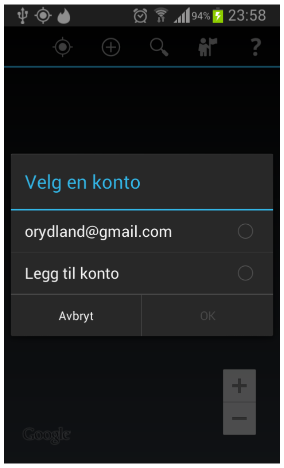


Figure 4.7: The choose account dialog

After the user has chosen an account, they are informed that the account has been created, and the map loads as shown in Figure 4.8.

The application will now start loading the list of artworks from the datastore in an AsyncTask, and mark each artwork with a marker as shown in Figure 4.9. This is also how the application will start when if the user has already chosen an account by running the application before.

The user can now zoom the map in and out by using the standard pinching movement on the screen and navigate the map by swiping in the direction they want the map to move. This can also be done while the artworks are loading, since the application performs the fetching in a background thread by delegating it to an AsyncTask.

Pressing a marker brings up the information fragment, as shown in Fig-

[4]`http://developer.android.com/reference/com/google/android/gms/common/AccountPicker.html`

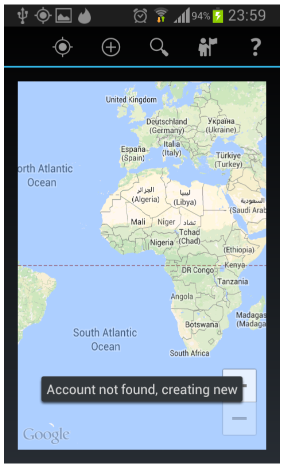


Figure 4.8: Map screen after account created

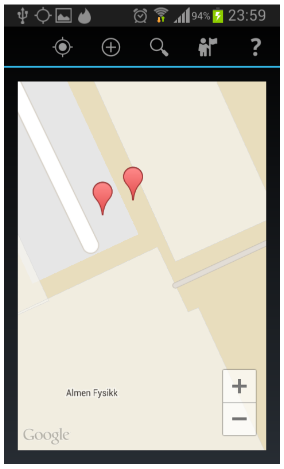


Figure 4.9: Markers on the map screen

ure 4.10 and centers the screen on the marker. This fragment only shows the name and short description of the artwork. The user can now press the information fragment to go to the artwork's information screen. To remove the fragment, the user can press anywhere outside the fragment. Pressing another marker will bring up that artwork's information screen instead, and center on the new artwork.

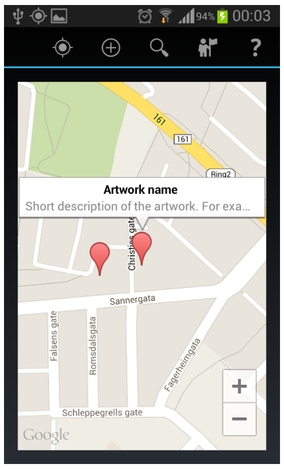
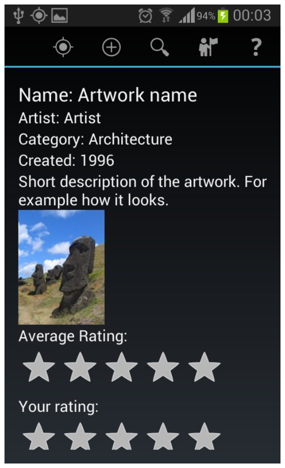
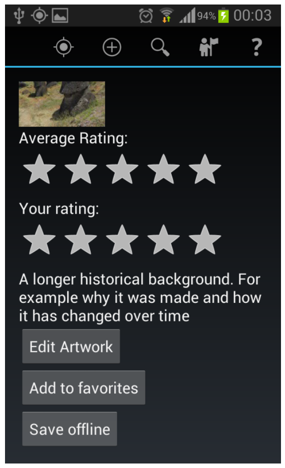


Figure 4.10: The information fragment

The artwork information screen When the user presses the information fragment, they are brought to the artwork's information screen. This screen is shown in Figure 4.11. Note that since the storage of pictures was not completed the picture shown is loaded locally from a static file on the device included in the project. It is meant to illustrate how the screen would look with a picture of the artwork included.


Name: Artwork name
Artist: Artist
Category: Architecture
Created: 1996
Short description of the artwork. For example how it looks.
Average Rating:
Your rating:

(a) The top of the artwork information page


Average Rating:
Your rating:
A longer historical background. For example why it was made and how it has changed over time

(b) The bottom of the artwork information page

Figure 4.11: Artwork information screen. The picture is a static picture loaded locally.

The screen is contained within a scrollview, so the user can scroll down the screen. The data is presented in the following order:

Name The name of the artwork.

Artist The artist that created the artwork.

Category The category of the artwork.

Year The year the artwork was created.

Short Description The short description of the artwork

Picture The picture of the artwork. As mentioned, not implemented, so picture is loaded from a static file on the device.

Average Rating This is the average rating stored in the database. This is implemented in the database and loaded from the database, but unfortunately the UI doesn't update correctly.

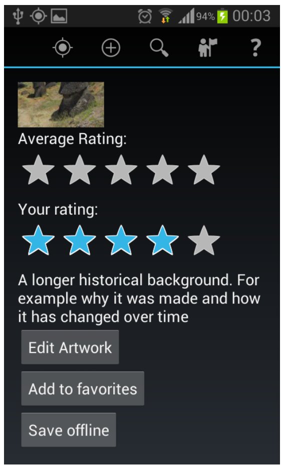
Your rating The rating given by the user to the artwork. This is not implemented completely either, so it doesn't update if the user has already rated the artwork previously, even though the data is fetched from the datastore. If the user gives the artwork a new rating, the old rating is replaced by the new one in the datastore, and the average rating in the datastore is updated. After the user has added a rating, the rating bar will be updated as shown in Figure 4.12.

Long description This is the longer, historical description of the artwork.

Edit Artwork This button is intended to take the user to a screen similar to the create artwork screen, but with the information of the artwork filled out in the fields. The user can then edit the fields and save the new information. Unfortunately this has not been implemented.

Add to favorites This will add the artwork to the user's list of favorites. This is done by adding the ID of the artwork to the favorites list of the UserAccount entity in the datastore. The user's chosen account is passed along with the ID to add the artwork to this user's favorites.

Add offline This is meant to add the artwork's information locally on the phone, so the user can access the information without being connected and having to download the information. But this functionality has not been implemented.



A longer historical background. For example why it was made and how it has changed over time

Figure 4.12: User adds rating

The add artwork screen When the user presses the add new button on the actionbar, they will be shown the dialog depicted in Figure 4.13

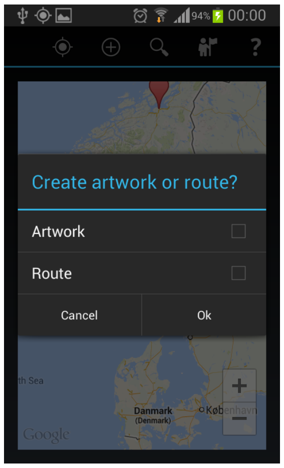


Figure 4.13: The dialog shown when user presses create new on the actionbar

By selecting "Artwork" and pressing "OK", the user will be brought to the add new artwork screen. The screen will initially be empty, as shown in Figure 4.14. Each field will contain a "hint", which is text that describes the intended content of the field, but disappears as soon as the user touches the field and starts typing.

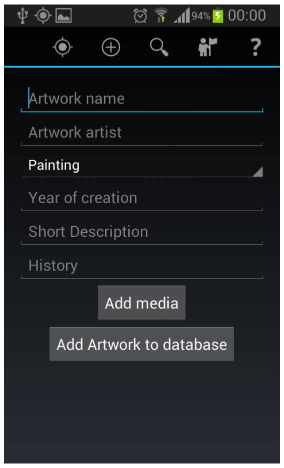


Figure 4.14: The empty create new artwork screen

When the user presses the Category field, a spinner[5] is shown, as seen in Fig-

[5]`http://developer.android.com/guide/topics/ui/controls/spinner.html`

ure 4.15. The spinner shows a set of pre-defined categories, currently painting, sculpture or architecture. These fields are a string-array defined in the projects res/values/strings folder. There is no validation in the datastore of the value of these fields.

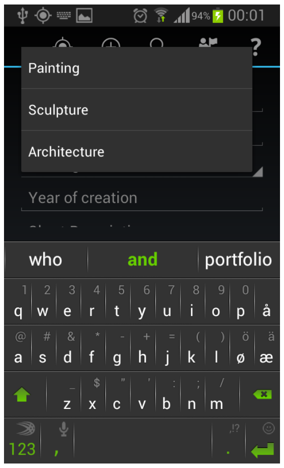
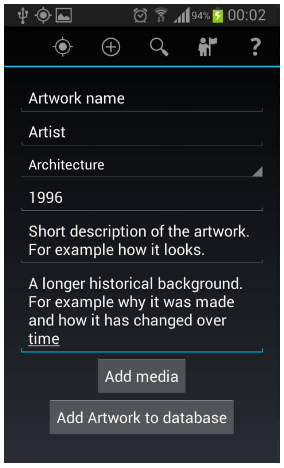


Figure 4.15: The category spinner

The year field will automatically open the numerical keyboard on the user's device, since the value of the input is defined as a number. This means that it is not actually possible to enter anything other than numbers into this field. However, the category and year field are the only fields that contain any kind of pre-defined data validation, so technically it is possible to leave all other fields empty and still save the artwork. When all the fields are filled out, the screen will look something like what is shown in Figure 4.16.

If the user presses the Add Media button, the application automatically launches the device's default camera app. The user can then take a picture, and choose to save it, or discard it. Should the user choose to discard the picture, the camera app will still continue running. The user will have to press the back button on their device to return to the create artwork screen. If the user chooses to save the picture they took, the picture will be shown on the create artwork screen as depicted in Figure 4.17.

When the user presses the Add artwork to database button, the application launches a background thread by using AsyncTask that passes the data to the database. However, as explained previously, the functionality to save media has not been implemented in the datastore, so the picture is simply discarded.


00:02

Artwork name

Artist

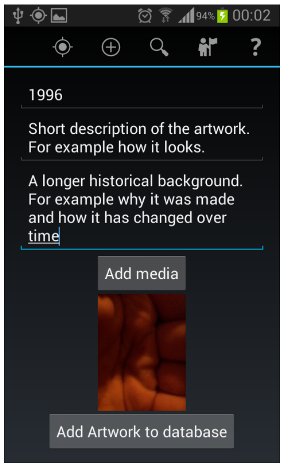
Architecture

1996

Short description of the artwork. For example how it looks.

A longer historical background. For example why it was made and how it has changed over time

Figure 4.16: Filled out new artwork screen


00:02

1996

Short description of the artwork. For example how it looks.

A longer historical background. For example why it was made and how it has changed over time

Figure 4.17: Picture added to the new artwork screen

The add new route screen Should the user instead choose to add a new route in the dialog shown in Figure 4.13, they will be brought to the add new route screen.

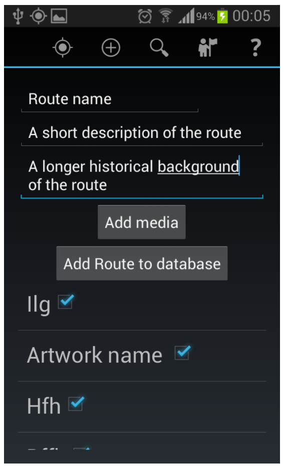

00:05
Route name
A short description of the route
A longer historical background of the route
Add media
Add Route to database
Ilg
Artwork name
Hfh

Figure 4.18: Adding a new route

The new route screen is somewhat similar to the add new artwork screen at first. It contains the following items:

Name The name the user wants to give the route

Short description A short description of the route

Long description Intended to fulfill the same goal as the long description/historical background of artworks.

Add Media Intended to function like the add media button in add artworks screen. But since the ability to store pictures in the datastore has not been implemented, it currently does nothing.

Add route to database This button adds the route to the database.

List of artworks Currently, this simple implementation of routes just list all the artworks available in the database. They are identified by name. Next to the name is a checkbox. If this checkbox is checked, the artwork's id is added to the route's list of artworks in the database when the route is stored.

Obviously, this is a very simple implementation. Users should be able to search for artworks by several fields, instead of remembering the name of artworks. It is also not possible for users to select the order of the artworks, as they will be stored in the order they appear in the list.

The list of routes screen The list of routes screen loads all the routes in the database and shows them in a list, as depicted in Figure 4.19.

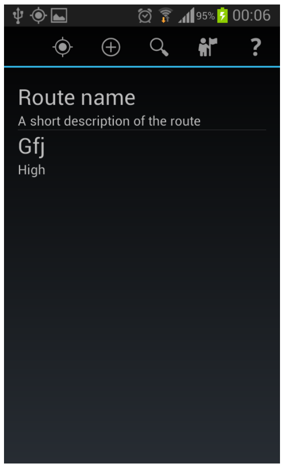


Figure 4.19: List of routes

The routes are listed by showing their name in large text and the description in smaller text. Ideally, there should be a way to see which artworks are included in the route, but this has not been implemented.

Map screen with routes When a user presses a route in the list of routes, they are brought to the map screen with the route drawn. Currently, the route is drawn in the order that the artworks appear in the list, which as mentioned previously, is the order that they appear in the add route list of artworks. In Figure 4.20 we see how the line between two artworks is drawn. The drawing uses Android Maps' addPolyLine[6] method.

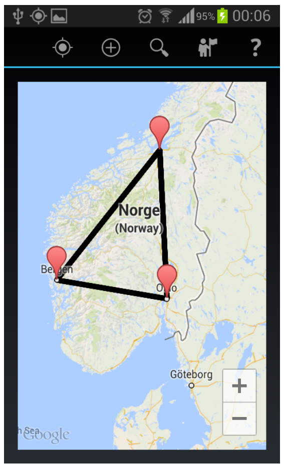


Figure 4.20: Routes

The problem with the ordering of artworks is illustrated in Figure 4.21

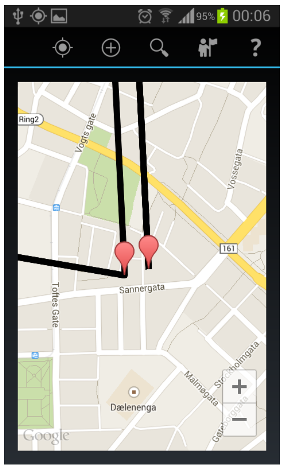


Figure 4.21: Routes order

[6]https://developers.google.com/maps/documentation/android/reference/com/google/android/gms/maps/model/Polyline

The favorites screen This screen is meant to list all the favorites the user has added to the list. The interface and the back-end has been implemented, but unfortunately the HTTP method required has not been finished. However, the code produced is included in Appendix B, in the FavoritesActivity.java class.

The top rated lists screen Unfortunately, this functionality was partially added to the back-end, but there was not enough time to start creating the user interface. The back-end code is included in Appendix B.

Helper classes

There are a few classes in the client that are not user interfaces, but still provide important functions in the application.

MyBaseActivity The MyBaseActivity class was created so every UI-class can inherit the actionbar. This makes it easy to make the actionbar consistent across all UI-screens.

The GeoLocationHelper The GeoLocationHelper class handles getting and updating the user's location using Android's location provider.

4.2 Tools and libraries

This section will explain the libraries and tools used in the development of this project in further detail.

4.2.1 Client

Android

The android framework provides several tools for creating mobile application that have been utilized in the client.

Android maps API

The android maps API allows developers to access many of Google Maps features inside an Android application. The API automatically handles gestures like panning and zooming. The API enables developers to add their own map markers and line segments to the map interface.

Activity

An Activity[7] is a window that the user can interact with. Within this window, users are presented with a full-screen interface. The user interface is defined using XML to define and position elements within the window, and called when

[7] `http://developer.android.com/reference/android/app/Activity.html`

the activity is created by calling setContentView(int)[8] in the activity's onCreate() method.

Every Android application needs to define a MainActivity, which acts as the startup activity for the application. In this application, this will be the MapActivity.

There are several more advanced functions provided by Android's Activity framework, but they are not used in this application.

AsyncTask

AsyncTask provides an easy way to perform background operations without having to deal with threading. By subclassing AsyncTask, the task is given a thread from Android's thread pool to execute the task. This allows users to still interact with the UI while the task is executing. In this application, the AsyncTask class is used to send and get data to and from the database without blocking the UI thread. When created, the task will automatically perform the doInBackground method of the task. When the task finishes, the result can be used by executing the onPostExecute method.

4.3 Back-end

While most of the code produced has been in the client, the most time consuming effort has definitely been the back-end and database. Structuring the database and relations among the objects proved more difficult than initially expected. Implementing an API for network traffic, was also more difficult than expected when custom methods had to be created.

4.3.1 Google App Engine

Google App Engine[9] allows developers to create web application that runs on Google's infrastructure. This provides automatic scaling with user requests. However, it does impose some limits too, especially on the structuring of data. This is because there is no guarantee that the data will exist on the same physical server.

4.3.2 Google App Engine Datastore

The datastore[10] is the database provided by the app engine framework. It is not a relational database like SQL-based database and similar. Instead it is a schemaless object datastore. This means that datastore objects are stored as *entities* that have one or more *properties*. A property is a name-value pair. The type of a property's value can be any of primitive types, the wrapper classes

[8]`http://developer.android.com/reference/android/app/Activity.html#setContentView(int)`

[9]`https://developers.google.com/appengine/?csw=1`

[10]`https://developers.google.com/appengine/docs/java/datastore/`

for the primitive datatypes, a String object or some Java collections, like Lists. Each entity is identified by it's *kind* and it's primary key. The primary key can either be a

4.3.3 Google Cloud Endpoints

Google Cloud Endpoints[11] is the tool used to generate the REST-API used to request and pass data over the internet between the server and the client. It is generated by recognizing the Objectify annotated code, and generating some basic API-methods to list all, get one, delete and edit the entity. For more advanced requests, the developer has to create their own methods.

4.3.4 Objectify

Objectify[12] is a library that simplifies querying the datastore. Initially, the web application utilized JDO to structure and query the data. However, JDO ended up limiting the of structure of lists of data. Objectify provides a lower-level access to the datastore. This allows developers more control over how they store and access the data.

[11] `https://developers.google.com/appengine/docs/java/endpoints/`
[12] `https://code.google.com/p/objectify-appengine/`

Chapter 5

Further work

5.1 Application

This section will detail some specific features that should be implemented to make this a complete application. It will also discuss tools that the literature and related application reviews have shown to be effective tools for crowdsourcing and geo-location applications.

5.1.1 Usability features

There are several smaller features needed to complete the application, especially regarding usability.

Search A search function is necessary for several reasons.

1. Assist users in finding artworks they are interested in.
2. Let users search for artworks to add to their routes. Obviously, listing all the artworks is not a scalable solution.
3. Users might want to find artworks they have heard about from other sources, either by name or some other parameter. Having to find this artwork on the map will be a chore.

To-do list A to-do list similar to the one in foursquare. The proposed recommender system could then calculate the most efficient way for a user to visit their to-do list. While users can save artworks they are interested in to their favorites, the favorite list is likely to also contain artworks they have visited.

Comments Users should be able to leave comments on artworks.

There should also be user testing and surveys conducted to receive feedback on the usability of the system and requested features from users.

5.1.2 Routes

There is still a lot of work to do regarding routes.

1. Users should be able to decide the order of the artworks in the route
2. Allow users to remove or add specific artworks in a route, and let the route be redrawn with the changes implemented
3. Display the total distance of a route

5.1.3 Efficiency

Both the performance of the application and the efficiency of the data transfer has been largely neglected during the development of the prototype application in this thesis. For a complete application, this will obviously have to be improved.

Performance is important for the user experience, as waiting for loading will frustrate users and means they will be less likely to continue using the application.

Data transfer efficiency is important for two reasons.

1. Users are likely to be on a limited data plan while using 3G network. This means they want the application to use as little data traffic as possible.
2. Limiting data traffic also means that all the tasks the application performs that are dependent on getting data from server are performed faster and the application appears more responsive to the user.

5.1.4 Offline storage and syncing

It would be possible to add a kind of local data storage on the user's phone. The user would have to select which artworks/routes they wanted to add to the local storage. This would allow users to download places they are interested in visiting from their home wi-fi before leaving. This feature will help further reduce the data load.

2 different ways of adding offline storage has been researched. Android provides the possibility of using a local SQLite database on the device. This would require mapping entities from the non-relational datastore to a SQL-based relational database. The other approach is using the SharedPreferences class. This behaves more similar to the key-value pairs provided by the AppEngine datastore, however, it is not how SharedPreferences is intended to be used.

5.1.5 Photos and albums

The ability to add photos to artworks/routes was not completed in this project. If this is added, there needs to be a way to organize the photos added to the artwork/route in an album. There also needs to be a way to determine what should be the primary picture of an artwork shown on the information fragment on the map screen.

5.1.6 Other media

Users might be interested in adding other media too, like audio clips or videos. There are currently not any similar android applications that offer this feature, possibly due to the size of audio/video files.

5.1.7 Search

The ability to search for artworks/routes has not been implemented during this thesis, but will obviously be necessary if the database gets somewhat large. Search criterias should include all the fields of the entities.

5.1.8 Further recommendation system

As shown in the literature study, being able to guide users to artworks they are likely interested in is very helpful. There are several criterias that could be employed in the recommender system:

Distance Like we saw in other similar applications, the user could select a maximum distance they are willing to travel, and all artworks or routes outside of this distance would be excluded.

Interests The user could could select certain categories they are interested in, and artworks of these categories would be higher weighted in the recommender system.

Rating The system could look at artworks or routes other users rate highly, and weigh these artworks or routes higher in the recommender system.

Metadata Other metadata could be connected to ratings, such as if users with similar interests rate an artwork or route highly, the item could be weighted higher for the first user too.

Social Media If social media is integrated into the application, routes or artworks rated highly by your friends could be weighted higher.

5.2 Navigation

One idea is to let the application implement a navigation system, similar to the one that was looked at in Waze. However, the navigation would have to be able to navigate by walking. Google Maps have a feature like this in public beta testing.

5.3 Gamification

Gamification means to add elements from games to other areas to increase motivation for employees or using an application. A very simple and typical example is giving users points for completing tasks, and having a high score lists so users can show off their score.

This is often heavily utilized in crowdsourcing applications to motivate users to provide information.

Gamification can also be used to improve the quality of information added by users. One could imagine allowing other users to rate the information a user has entered. This way people would strive to enter correct information to achieve a high rating.

In the literature review studies showed that while gamification is an effective tool for user motivation, it must be designed correctly.

5.4 Web interface

As seen in the research of the related applications, many applications that wish to assist users in finding places, and rely on users providing information about geographical locations utilize a web interface in addition to a geo-location mobile application.

There are several advantages the addition of a web interface would provide to the PubArt application.

In the literature review studies showed the effect discussion pages had on the quality of content on Wikipedia. A web interface would allow users to discuss changes to artwork and route information pages. A web interface would also make it easier to add more thorough information about an artwork, as typing extensive information could prove difficult on mobile devices. It would also be simpler to include links to other artworks from the information pages.

5.5 Administration tools

One of the main challenges of crowdsourcing is validating the information users add or edit. For this some sort of administration tools needs to be implemented for the application. The best way to do this would probably be to integrate the administration tools with a web-interface.

There are different ways of implementing administration moderation of information. One could not let any addition/edit users make take effect until an administrator approve it. The other way would be to let the change take effect immediately, but keep a backup of older versions of the information, and allow administrators to rollback any changes made.

5.6 Social Media

Several of the related applications that have been researched in this thesis contain elements of social media and/or are integrated with existing social media like facebook or twitter.

Especially allowing content from the application to be shared to other social media like facebook and twitter would be a very effective tool, both to engage users and to assist in getting new users to try the application.

Chapter 6

Results and discussion

This section will discuss the result of the thesis.

6.1 Results

6.1.1 Research questions

How can we motivate users to use and contribute to the application?

Through our literature study and review of relevant applications, we have seen examples of different ways to motivate users to contribute. One way we have seen used in several applications and discussed in literature is using gamification elements to motivate users. However, some care must be taken when using gamification. It is important that the game is not more important than the information added by the users.

Are there other concepts than the ones mentioned that could be used to improve this application?

In our study of Wikipedia, we saw that one of the most important ways to motivate users to add quality content is to instill a sense of community and ownership of the product they produce. Considering our application is more about sharing knowledge than the other applications studied, it is important to consider ways to validate the information added.

What can we do to help users find the art they are interested in?

We have seen that the other popular crowdsourcing applications utilizes social networking, recommendation systems, ratings and comments to assist users in discovering interesting places.

What can we learn from other similar applications that can be utilized in this application?

As already mentioned, gamification is an element heavily utilized in crowdsourcing applications, and seems to be an effective way to motivate users to contribute and use the application.

Other discoveries

We have seen that there is much research into incorporating mobile devices to provide interactive learning experiences. We could definitely incorporate features into this application that would support such a goal.

6.1.2 Project goals

Allow users to add information about artworks to a database

The ability to add information was already added during the depth study, however, the functionality was extended during this thesis. Users are now uniquely identified by their chosen Google account. The functionality to add media has been implemented in the client, but can not be stored in the database.

Allow users to get information added by other users

This was also added during the depth study, but the presentation of the data has been significantly approved.

Allow users to add to or edit information already added to the database

The ability to edit information has not been added, but users can now rate the artworks and routes in the database, and the average rating of the items will be updated.

Allow users to create routes of several different artworks

The ability to create routes has been implemented, but the functionality is very limited.

Assist users in finding artworks or routes they are likely to be interested in

While a recommendation system has not been added, some of the features required has been started. By letting users rate content it is possible to extract metadata about some features the artwork they like share. The ability to calculate average ratings also means that users can be shown

6.2 Discussion

6.2.1 Limitations

The results of the project is naturally limited by the fact that both the research and development has been conducted by a single person. The goals of the thesis has been quite broad, which means that some of the research conducted has been more superficial than it should have been.

Due to limited time, only 4 related applications were studied in detail. There are many more crowdsourcing platforms out there, that employ a huge variety of methods in motivating users and assuring the quality of the content provided.

The thesis originally planned to conduct user surveys as more features were implemented would have provided a broader range of ideas and valuable feedback to limit the drawback of bias implicated by having a single person responsible for the research and specifications.

The research conducted by the project would have benefited from a more focused research area. The research tried to look into many different ways of employing the application for different goals. It would probably have been a good idea to specify a clearer goal of the research, whether into crowdsourcing and the related problems, learning and the way the application could support it or gamification and similar concepts to motivate users.

6.2.2 Development methodology

One issue with the development of the application has been that the goals and requirements were not defined clearly enough. The functionality should have been specified further. Even though the idea of an iterative approach is good, especially for a single developer, it can be difficult to keep moving forwards when critical components of the application prove harder to implement than expected.

The development part of the thesis should have conducted a more thorough research into possible libraries and frameworks to utilize in the development of the project. For example, if the project had used the objectify datastore library from the beginning, a lot of time and frustration would have been saved.

Had the implementation gotten further, letting users test the application would have given valuable feedback to the usability features.

Chapter 7

Conclusion

Through the study of related applications we have seen that there are several successful mobile apps that rely on users providing knowledge and information through crowdsourcing. However, our study has also shown that it is hard to motivate users to provide this information for free, and that validating the information provided is a major challenge.

The combination of our literature study and review of other applications we have seen that the use of interactive mobile devices like PDAs and producing trails in museums has been attempted, and that the idea of lists of items in crowdsourcing applications has been made, but the two have not been combined into an interactive learning experience.

Unfortunately this thesis hasn't implemented all the features planned. However, it has shown that the concept of a crowdsourcing application that can be used to enable learning about public art is definitely valid. It has been shown that by combining features like gamification and social network of popular applications like FourSquare and Waze with the quality assurance of a crowdsourcing encyclopedia like Wikipedia we could fulfill the need for an interactive learning experience about art in public places. The thesis suggests further work to complete the application, and possible ways to motivate users to add their knowledge and provide quality assurance to the information gathered.

Bibliography

[1] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[2] M. Haklay, "How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets," *Environment and planning. B, Planning & design*, vol. 37, no. 4, p. 682, 2010.

[3] M. Hu, E.-P. Lim, A. Sun, H. W. Lauw, and B.-Q. Vuong, "Measuring article quality in wikipedia: models and evaluation," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pp. 243–252, ACM, 2007.

[4] B. Stvilia, M. B. Twidale, L. Gasser, and L. C. Smith, "Information quality discussions in wikipedia," in *Proceedings of the 2005 international conference on knowledge management*, pp. 101–113, O'Reilly, 2005.

[5] A. Kittur, E. Chi, B. A. Pendleton, B. Suh, and T. Mytkowicz, "Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie," *World Wide Web*, vol. 1, no. 2, p. 19, 2007.

[6] D. Chandler and A. Kapelner, "Breaking monotony with meaning: Motivation in crowdsourcing markets," *Journal of Economic Behavior & Organization*, 2013.

[7] J. Rogstadius, V. Kostakos, A. Kittur, B. Smus, J. Laredo, and M. Vukovic, "An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets.," *ICWSM*, 2011.

[8] C. Eickhoff, C. G. Harris, A. P. de Vries, and P. Srinivasan, "Quality through flow and immersion: gamifying crowdsourced relevance assessments," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pp. 871–880, ACM, 2012.

[9] T. Y. Lee, C. Dugan, W. Geyer, T. Ratchford, J. Rasmussen, N. S. Shami, and S. Lupushor, "Experiments on motivational feedback for crowdsourced workers," in *Seventh International AAAI Conference on Weblogs and Social Media*, 2013.

[10] L. Naismith, M. Sharples, G. Vavoula, and P. Lonsdale, “Literature review in mobile technologies and learning,” 2004.

[11] R. Reynolds, K. Walker, and C. Speight, “Web-based museum trails on pdas for university-level design students: Design and evaluation,” *Computers & Education*, vol. 55, no. 3, pp. 994–1003, 2010.

[12] G. Vavoula, M. Sharples, P. Rudman, J. Meek, and P. Lonsdale, “Myartspace: Design and evaluation of support for learning with multimedia phones between classrooms and museums,” *Computers & Education*, vol. 53, no. 2, pp. 286–299, 2009.

Appendix A

Installation Guide

The application has mainly been tested on a Samsung Galaxy S2 device running Android 4.1.2 and a Samsung Galaxy S4 running Android 4.2.2. However, the application should be fully functional on any device running Android 3.0 or newer. It will not work for any older versions of Android.

To run the application on an android device it is necessary to download and install some specific software. The Eclipse IDE is required to be able to run the code on android devices. Eclipse can be downloaded here: `http://www.eclipse.org/`.

When Eclipse is installed, the *Android Development Tools* [1] plugin needs to be installed. To install it follow these steps:

1. Select the "Help" menu in eclipse.
2. Select "Install new software".
3. Click "Add" in the top-right corner.
4. In the Add Repository dialog, enter "ADT Plugin" for the *Name* and the following url for *Location*: https://dl-ssl.google.com/android/eclipse/
5. Click "OK".
6. Select checkbox nect to "Developer Tools" and click "Next".
7. A list of tools appears, click "Next".
8. Read and accept the license agreements, click "Finish".
9. When the installation is complete, restart eclipse.

[1]`http://developer.android.com/sdk/installing/installing-adt.html`

After this, right click inside the package explorer in eclipse, choose import, select from archive file and select the .zip file delivered with the thesis. The import should include everything, like classpaths, folder structure, etc.

Now, connect the Android Device to the computer with a USB cable. When connected, run the project and select your device as the target device. The application should start automatically. The application will connect to the database illustrated in this project.

However, using the Google plugin for eclipse, it is also possible to deploy the application on your own to see how it works. To do this, install the Google plugin for eclipse as described here: `https://developers.google.com/eclipse/`.

When this has been installed, create a web application here: `https://appengine.google.com/`. When you have been given an application ID, right click the PubArt_AppEngine project, select Google -¿ App Engine settings. Enter the application ID into the application ID field and click OK. Right click the project again and select Deploy to App Engine. Press Deploy. If you now go to `https://appengine.google.com` the project should appear, and if you click on it you will be brought to the application's developer console. More information is available here: `https://developers.google.com/eclipse/docs/getting_started`

Appendix B

Produced Source Code

It's hard to separate the code produced during the depth study and this thesis as some features during the thesis was added to the existing classes. Therefore this appendix contains all the code produced for entire project.

B.1 Android Client

```
package com.example.pubart;

public class AddArtworkActivity extends MyBaseActivity {
	private static final int TAKE_PICTURE_REQUESTCODE = 1112;
	private static final String TAG = "AddArtworkActivity";

	private EditText etName;
	private EditText etArtist;
	private Spinner categorySpinner;
	private EditText etYear;
	private EditText etTextualDescription;
	private EditText etTextualBackground;

	private ImageView artworkPhoto;
	private GeoLocationHelper geoLocationHelper = new GeoLocationHelper();

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		// Set contenview to XML file
		setContentView(R.layout.activity_add_artwork);

		findViewsById();

		geoLocationHelper.startRetrievingLocation(this);

	}

	public void startCameraButton(View view) {
		dispatchTakePictureIntent(TAKE_PICTURE_REQUESTCODE);
```

```
    }

    private void dispatchTakePictureIntent(int actionCode) {
            Intent takePictureIntent = new Intent(MediaStore.
                 ACTION_IMAGE_CAPTURE);
            startActivityForResult(takePictureIntent, actionCode);
    }

    private void handleSmallCameraPhoto(Intent intent) {
            Bundle extras = intent.getExtras();
            artworkPhoto.setImageBitmap((Bitmap) extras.get("data"));
            artworkPhoto.setScaleType(ImageView.ScaleType.FIT_CENTER);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
         data) {
            if (requestCode == TAKE_PICTURE_REQUESTCODE && resultCode ==
                 RESULT_OK) {
                    handleSmallCameraPhoto(data);
            }
    }

    public void addArtworkButton(View view) {
            new AddNewArtworkTask().execute(getApplicationContext());
            Toast.makeText(getApplicationContext(), "Artwork added to database
                 ",
                            Toast.LENGTH_LONG).show();
    }

    private class AddNewArtworkTask extends AsyncTask<Context, Void, Long> {

            protected Long doInBackground(Context... contexts) {

                    Artworkendpoint.Builder artworkEndpointBuilder = new
                         Artworkendpoint.Builder(
                                    AndroidHttp.newCompatibleTransport(), new
                                         JacksonFactory(),
                                    new HttpRequestInitializer() {

                                            @Override
                                            public void initialize(HttpRequest
                                                 httpRequest) {

                                            }
                                    });
                    Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                         updateBuilder(
                                    artworkEndpointBuilder).build();

                    try {
                            ArtWork artwork = new ArtWork();
                            artwork.setName(etName.getText().toString());
                            artwork.setArtist(etArtist.getText().toString());
                            artwork.setCategory(categorySpinner.getSelectedItem
                                 ()
                                            .toString());
```

```
                        artwork.setYear(etYear.getText().toString());
                        artwork.setTextualBackground(etTextualBackground.
                            getText()
                                    .toString());
                        artwork.setTextualDescription(etTextualDescription.
                            getText()
                                    .toString());
                        artwork.setLatitude(geoLocationHelper.
                            getCurrentLocation()
                                    .getLatitude());
                        artwork.setLongitude(geoLocationHelper.
                            getCurrentLocation()
                                    .getLongitude());
                        artwork.setAverageRating(0);
                        ArtWork result = artworkEndpoint.insertArtWork(
                            artwork)
                                    .execute();

                } catch (IOException IOE) {
                        IOE.printStackTrace();
                        Log.e(TAG, "IO exception on doInBackground", IOE);
                }
                return (long) 0;
        }

        protected void onPostExectue() {
                Toast.makeText(getApplicationContext(),
                                "Artwork added to database", Toast.
                                    LENGTH_LONG).show();
        }

    }

    private void findViewsById() {
        etName = (EditText) findViewById(R.id.editTextName);
        etArtist = (EditText) findViewById(R.id.editTextArtist);
        categorySpinner = (Spinner) findViewById(R.id.spinnerCategory);
        ArrayAdapter<CharSequence> spinnerAdapter = ArrayAdapter
                        .createFromResource(this, R.array.categories_array,
                                    android.R.layout.simple_spinner_item)
                                        ;
        spinnerAdapter
                        .setDropDownViewResource(android.R.layout.
                            simple_spinner_dropdown_item);
        categorySpinner.setAdapter(spinnerAdapter);
        etYear = (EditText) findViewById(R.id.editTextYear);
        etTextualDescription = (EditText) findViewById(R.id.
            editTextDescription);
        etTextualBackground = (EditText) findViewById(R.id.
            editTextBackground);

        artworkPhoto = (ImageView) findViewById(R.id.ivArtworkPhoto);
    }

}
```

```
package com.example.pubart;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.example.pubart.artworkendpoint.Artworkendpoint;
import com.example.pubart.artworkendpoint.model.ArtWork;
import com.example.pubart.artworkrouteendpoint.Artworkrouteendpoint;
import com.example.pubart.artworkrouteendpoint.model.ArtworkRoute;
import com.example.pubart.helpers.MyBaseActivity;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson.JacksonFactory;

public class AddRouteActivity extends MyBaseActivity {

	private static final String TAG = "addrouteactivity";
	ListView lv;
	ArrayList<ListItem> artworksToList;
	private SharedPreferences preferences;
	private String accountName;
	EditText etName, etDescription, etBackground;
	ArrayList<Long> artworksToAdd;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_add_route);

		preferences = getSharedPreferences(PREFS_NAME, 0);
		if (preferences.getString("ACCOUNT_ID", "None").equals("None")) {
			Toast.makeText(getApplicationContext(), "Could not get
			     account",
					Toast.LENGTH_SHORT).show();
		} else {
```

```
                accountName = preferences.getString("ACCOUNT_ID", "No
                    account");
            }

            etName = (EditText) findViewById(R.id.etRouteName);
            etDescription = (EditText) findViewById(R.id.etRouteDescription);
            etBackground = (EditText) findViewById(R.id.etRouteBackground);

            artworksToAdd = new ArrayList<Long>();
            artworksToList = new ArrayList<ListItem>();

            new getArtworksTask().execute();
    }

    private void populateListView(ArrayList<ListItem> items) {
            ArrayAdapterItem adapter = new ArrayAdapterItem(this,
                            R.layout.list_view_one_name, items);

            ListView listViewItems = (ListView) findViewById(R.id.artworkList)
                 ;
            listViewItems.setAdapter(adapter);
            listViewItems
                            .setOnItemClickListener(new
                                onItemClickListenerListViewItem());
    }

    public void addRouteButton(View view) {
            new AddNewRouteTask().execute(getApplicationContext());
            Toast.makeText(getApplicationContext(), "Route added to database",
                            Toast.LENGTH_LONG).show();
    }

    private class getArtworksTask extends AsyncTask<Void, Void, List<ArtWork
         >> {

            @Override
            protected void onPostExecute(List<ArtWork> artworks) {
                    if (artworks == null || artworks.isEmpty()) {
                            Toast.makeText(getApplicationContext(),
                                            "Failed to load artworks", Toast.
                                                LENGTH_SHORT).show();
                    } else {
                            for (ArtWork artwork : artworks) {
                                    artworksToList.add(new ListItem(artwork.
                                        getName(), artwork
                                                .getId()));
                            }
                            populateListView(artworksToList);
                    }
            }

            @Override
            protected List<ArtWork> doInBackground(Void... parms) {

                    List<ArtWork> artworks = new ArrayList<ArtWork>();
```

```
                        Artworkendpoint.Builder artworkEndpointBuilder = new
                             Artworkendpoint.Builder(
                                        AndroidHttp.newCompatibleTransport(), new
                                             JacksonFactory(),
                                        new HttpRequestInitializer() {

                                                @Override
                                                public void initialize(HttpRequest
                                                     httpRequest) {

                                                }
                                        });
                        Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                             updateBuilder(
                                        artworkEndpointBuilder).build();

                        try {
                                artworks = artworkEndpoint.listArtWork().execute().
                                     getItems();
                                Log.d(TAG, "getting artwork");
                                return artworks;
                        } catch (Exception exception) {
                                exception.printStackTrace();
                                Log.e(TAG, "IOException on doInBackground",
                                     exception);
                        }

                        return null;
                }
        }

        class ArrayAdapterItem extends ArrayAdapter<ListItem> {

                Context mContext;
                int layoutResourceId;
                ArrayList<ListItem> data;

                public ArrayAdapterItem(Context mContext, int layoutResourceId,
                                ArrayList<ListItem> data) {

                        super(mContext, layoutResourceId, data);

                        this.layoutResourceId = layoutResourceId;
                        this.mContext = mContext;
                        this.data = data;
                }

                @Override
                public View getView(int position, View convertView, ViewGroup
                     parent) {

                        if (convertView == null) {
                                LayoutInflater inflater = ((Activity) mContext)
                                                .getLayoutInflater();
                                convertView = inflater.inflate(layoutResourceId,
                                     parent, false);
                        }
                CheckBox lChk = ((CheckBox) convertView.findViewById(R.id.
                     checkBoxItem));

                ListItem listItem = data.get(position);

                final TextView textViewItem = (TextView) convertView
                                .findViewById(R.id.textViewItem);
                textViewItem.setText(listItem.getName());
                textViewItem.setTag(listItem.getId());



                lChk.setOnCheckedChangeListener(new OnCheckedChangeListener
                     () {

                        @Override
                        public void onCheckedChanged(CompoundButton
                             buttonView, boolean isChecked) {
                                if (isChecked) {
                                        artworksToAdd.add(Long.parseLong(
                                             textViewItem.getTag().toString())
                                             );
                                } else {
                                        artworksToAdd.remove(Long.parseLong(
                                             textViewItem.getTag().toString())
                                             );
                                }
                        }
                });

                return convertView;
        }
}

class onItemClickListenerListViewItem implements OnItemClickListener {

        @Override
        public void onItemClick(AdapterView<?> parent, View view, int
             position,
                        long id) {

                Context context = view.getContext();

                TextView textViewItem = ((TextView) view
                                .findViewById(R.id.textViewItem));
                CheckBox checkBoxItem = ((CheckBox) view
                                .findViewById(R.id.checkBoxItem));
                String listItemText = textViewItem.getText().toString();
                Long listItemId = Long.parseLong(textViewItem.getTag().
                     toString());
                boolean checkBoxStatus = checkBoxItem.isChecked();

                if (checkBoxStatus) {
                        Toast.makeText(context,
                                        "Item: " + listItemText + ", Item ID:
                                              " + listItemId,
                                        Toast.LENGTH_SHORT).show();
                    }
            }

    }

    class ListItem {
            String name;
            Long id;

            public ListItem(String name, Long id) {
                    this.name = name;
                    this.id = id;
            }

            public String getName() {
                    return name;
            }

            public Long getId() {
                    return id;
            }
    }

    private class AddNewRouteTask extends AsyncTask<Context, Void, Long> {

            ArrayList<Long> listOfArtworks;

            protected Long doInBackground(Context... contexts) {

                    Artworkrouteendpoint.Builder artworkRouteEndpointBuilder =
                         new Artworkrouteendpoint.Builder(
                                    AndroidHttp.newCompatibleTransport(), new
                                         JacksonFactory(),
                                    new HttpRequestInitializer() {

                                            @Override
                                            public void initialize(HttpRequest
                                                 httpRequest) {

                                            }
                                    });
                    Artworkrouteendpoint artworkRouteEndpoint =
                         CloudEndpointUtils
                                    .updateBuilder(artworkRouteEndpointBuilder).
                                         build();

                    try {
                            ArtworkRoute artworkRoute = new ArtworkRoute();
                            artworkRoute.setName(etName.getText().toString());
                            artworkRoute.setDescription(etDescription.getText().
                                 toString());
                            artworkRoute.setBackground(etBackground.getText().
                                 toString());
                            artworkRoute.setArtworks(artworksToAdd);
```

```
                        ArtworkRoute result = artworkRouteEndpoint.
                            insertArtworkRoute(
                                    artworkRoute).execute();

                    } catch (IOException IOE) {
                        IOE.printStackTrace();
                        Log.e(TAG, "IO exception on doInBackground", IOE);
                    }
                    return (long) 0;
            }

        }

}
```

```
package com.example.pubart;

import java.util.Map;

public class ArtworkEntity {

        private String name, shortDescription, longDescription, year, artist,
            category;
        private Double latitude, longitude;
        private Map<String, Integer> ratings;
        private Integer averageRating;
        private Long id;

        public ArtworkEntity(Long id, String name, String shortDescription,
            String longDescription, String year, String artist, String category,
            Double latitude, Double longitude, Integer averageRating) {
                this.id = id;
                this.name = name;
                this.shortDescription = shortDescription;
                this.longDescription = longDescription;
                this.year = year;
                this.artist = artist;
                this.category = category;
                this.latitude = latitude;
                this.longitude = longitude;
                this.averageRating = averageRating;
        }

        public Long getId() {
                return id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
```

```
	public String getShortDescription() {
		return shortDescription;
	}

	public void setShortDescription(String shortDescription) {
		this.shortDescription = shortDescription;
	}

	public String getLongDescription() {
		return longDescription;
	}

	public void setLongDescription(String longDescription) {
		this.longDescription = longDescription;
	}

	public String getYear() {
		return year;
	}

	public void setYear(String year) {
		this.year = year;
	}

	public String getArtist() {
		return artist;
	}

	public void setArtist(String artist) {
		this.artist = artist;
	}

	public String getCategory() {
		return category;
	}

	public void setCategory(String category) {
		this.category = category;
	}

	public Double getLatitude() {
		return latitude;
	}

	public void setLatitude(Double latitude) {
		this.latitude = latitude;
	}

	public Double getLongitude() {
		return longitude;
	}

	public void setLongitude(Double longitude) {
		this.longitude = longitude;
	}

	public void setRatings(Map<String, Integer> ratings) {
		this.ratings = ratings;
	}

	public Map<String, Integer> getRatings() {
		return ratings;
	}

}
```

```
package com.example.pubart;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.TextView;
import android.widget.Toast;

import com.example.pubart.artworkendpoint.Artworkendpoint;
import com.example.pubart.artworkendpoint.model.ArtWork;
import com.example.pubart.artworkendpoint.model.IntegerCollection;
import com.example.pubart.helpers.MyBaseActivity;
import com.example.pubart.useraccountendpoint.Useraccountendpoint;
import com.example.pubart.useraccountendpoint.model.UserAccount;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson.JacksonFactory;

public class ArtworkInformationActivity extends MyBaseActivity implements
		RatingBar.OnRatingBarChangeListener {

	private static final String TAG = "ArtworkInformationActivity";
	Long artworkId;
	String accountName;
	ArtWork artwork;

	TextView tvName, tvCategory, tvArtist, tvYearOfCreation,
			tvShortDescription, tvHistory, tvNameLabel, tvArtistLabel,
				tvYearLabel, tvCategoryLabel;
	ImageView ivArtworkPhoto;
	RatingBar rbAverageRating, rbUserRating;
	Button bAddFavorite;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_artwork_information);
		findViewsById();
```

```
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            artworkId = extras.getLong("ARTWORK_ID");
            accountName = extras.getString("ACCOUNT_ID");
        }
        new getArtworksTask().execute();
        rbUserRating.setOnRatingBarChangeListener(this);

    }

    public void addFavoriteButton(View view) {
        new addFavoriteTask().execute();
    }

    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating,
            boolean fromUser) {
        Integer intRating = Math.round(rating);
        new insertRatingTask().execute(intRating);

    }


    private void findViewsById() {
        tvName = (TextView) findViewById(R.id.tvShowArtWorkName);
        tvArtist = (TextView) findViewById(R.id.tvShowArtistName);
        tvCategory = (TextView) findViewById(R.id.tvShowCategory);
        tvYearOfCreation = (TextView) findViewById(R.id.
             tvShowYearOfCreation);
        tvShortDescription = (TextView) findViewById(R.id.
             tvShowShortDescription);
        tvHistory = (TextView) findViewById(R.id.tvShowHistory);
        ivArtworkPhoto = (ImageView) findViewById(R.id.ivArtworkPhoto);
        ivArtworkPhoto.setImageResource(R.drawable.artwork);
        rbAverageRating = (RatingBar) findViewById(R.id.rbAverageRating);
        rbUserRating = (RatingBar) findViewById(R.id.rbUserRating);
        bAddFavorite = (Button) findViewById(R.id.bAddToFavorites);
        tvNameLabel = (TextView) findViewById(R.id.tvNameLabel);
        tvArtistLabel = (TextView) findViewById(R.id.tvArtistLabel);
        tvCategoryLabel = (TextView) findViewById(R.id.tvCategoryLabel);
        tvYearLabel = (TextView) findViewById(R.id.tvYearLabel);
    }

    private class getArtworksTask extends AsyncTask<Void, Void, ArtWork> {

        @Override
        protected ArtWork doInBackground(Void... params) {

            Artworkendpoint.Builder artworkEndpointBuilder = new
                 Artworkendpoint.Builder(
                         AndroidHttp.newCompatibleTransport(), new
                              JacksonFactory(),
                         new HttpRequestInitializer() {

                             @Override
                             public void initialize(HttpRequest
                                  httpRequest) {
                                }
                        });
                Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                    updateBuilder(
                            artworkEndpointBuilder).build();

                try {
                        artwork = artworkEndpoint.getArtWork(artworkId).
                            execute();
                        return artwork;
                } catch (IOException e) {
                        Log.e(TAG, "IO exception on doInBackground", e);
                        e.printStackTrace();
                }

                return null;
        }

        @Override
        protected void onPostExecute(ArtWork artwork) {
                Log.d(TAG, "artwork is " + artworkId);
                if (artwork == null) {
                        Toast.makeText(getApplicationContext(),
                                        "Could not get artwork", Toast.
                                            LENGTH_SHORT).show();
                } else {
                        tvName.setText(artwork.getName());
                        tvArtist.setText(artwork.getArtist());
                        tvCategory.setText(artwork.getCategory());
                        tvYearOfCreation.setText(artwork.getYear());
                        tvShortDescription.setText(artwork.
                            getTextualDescription());
                        tvHistory.setText(artwork.getTextualBackground());
                        rbAverageRating.setNumStars(artwork.getAverageRating
                            ());
                        // new getUserRatingTask().execute();
                }
        }

}

private class insertRatingTask extends AsyncTask<Integer, Void, Void> {

        @Override
        protected Void doInBackground(Integer... params) {

                Artworkendpoint.Builder artworkEndpointBuilder = new
                    Artworkendpoint.Builder(
                            AndroidHttp.newCompatibleTransport(), new
                                JacksonFactory(),
                            new HttpRequestInitializer() {

                                    @Override
                                    public void initialize(HttpRequest
                                        httpRequest) {
                                        }
                                });
                        Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                             updateBuilder(
                                        artworkEndpointBuilder).build();

                        try {
                                artworkEndpoint.insertUserRating(params[0],
                                     accountName,
                                                artwork).execute();
                        } catch (IOException e) {
                                Log.e(TAG, "IO exception on doInBackground", e);
                                e.printStackTrace();
                        }

                        return null;
                }
        }

        private class addFavoriteTask extends AsyncTask<Void, Void, Void> {

                @Override
                protected Void doInBackground(Void... params) {
                        Useraccountendpoint.Builder userAccountEndpointBuilder =
                             new Useraccountendpoint.Builder(
                                        AndroidHttp.newCompatibleTransport(), new
                                             JacksonFactory(),
                                        new HttpRequestInitializer() {

                                                @Override
                                                public void initialize(HttpRequest
                                                     httpRequest) {

                                                }

                                        });
                        Useraccountendpoint userAccountEndpoint =
                             CloudEndpointUtils
                                        .updateBuilder(userAccountEndpointBuilder).
                                             build();

                        try {
                                userAccountEndpoint.addFavorite(accountName,
                                     artworkId).execute();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        return null;
                }

        }


}
```

```
package com.example.pubart;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.content.Context;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import com.example.pubart.artworkendpoint.Artworkendpoint;
import com.example.pubart.helpers.MyBaseActivity;
import com.example.pubart.useraccountendpoint.Useraccountendpoint;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson2.JacksonFactory;

@SuppressLint("NewApi")
public class FavoritesAcitivity extends MyBaseActivity {

        private ArrayList<String> values;
        private List<Long> ids;
        private String accountName;
        ListView lv;

        @SuppressLint("NewApi")
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                setContentView(R.layout.acitivity_favorites);
                super.onCreate(savedInstanceState);

                Bundle extras = getIntent().getExtras();
                if (extras != null) {
                        accountName = extras.getString("ACCOUNT_ID");
                }

                lv = (ListView) findViewById(R.id.listview);

                values = new ArrayList<String>();
                ids = new ArrayList<Long>();

                new getFavoritesTask().execute();

        }

        private void populateFavoritesList(boolean result) {
                if (result) {
                        final ArrayList<String> list = new ArrayList<String>();
```

```
                    for (int i = 0; i < values.size(); i++) {
                          list.add(values.get(i));
                    }

                    final StableArrayAdapter stableArrayAdapter = new
                         StableArrayAdapter(
                                   this, android.R.layout.
                                        simple_expandable_list_item_1, list);
                    lv.setAdapter(stableArrayAdapter);

                    lv.setOnItemClickListener(new AdapterView.
                         OnItemClickListener() {

                          @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR1)
                          @SuppressLint("NewApi")
                          @Override
                          public void onItemClick(AdapterView<?> parent, final
                                View view,
                                         int position, long id) {
                                final String item = (String) parent
                                               .getItemAtPosition(position);
                                view.animate().setDuration(2000).alpha(0)
                                               .withEndAction(new Runnable()
                                                    {

                                                       @Override
                                                       public void run() {

                                                              list.remove(item
                                                                   );
                                                              stableArrayAdapter
                                                                   .
                                                                   notifyDataSetChanged
                                                                   ();
                                                              view.setAlpha(1)
                                                                   ;

                                                       }
                                               });

                          }

                    });
             } else {
                    Toast.makeText(getApplicationContext(), "Could not get
                         favorites",
                                   Toast.LENGTH_LONG).show();
             }
      }

      private class StableArrayAdapter extends ArrayAdapter<String> {

             HashMap<String, Integer> mIdMap = new HashMap<String, Integer>();

             public StableArrayAdapter(Context context, int textViewResourceId,
                          List<String> objects) {
                    super(context, textViewResourceId, objects);
```

```
                        for (int i = 0; i < objects.size(); i++) {
                                mIdMap.put(objects.get(i), i);
                        }
                }

                @Override
                public long getItemId(int position) {
                        String item = getItem(position);
                        return mIdMap.get(item);
                }

                @Override
                public boolean hasStableIds() {
                        return true;
                }

        }

        private class getFavoriteNamesTask extends
                        AsyncTask<List<Long>, String, List<String>> {

                @Override
                protected ArrayList<String> doInBackground(List<Long>... params) {

                        List<String> strings = new ArrayList<String>();

                        Artworkendpoint.Builder artworkEndpointBuilder = new
                             Artworkendpoint.Builder(
                                        AndroidHttp.newCompatibleTransport(), new
                                             JacksonFactory(),
                                        new HttpRequestInitializer() {

                                                @Override
                                                public void initialize(HttpRequest
                                                     httpRequest) {

                                                }
                                        });

                        Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                             updateBuilder(
                                        artworkEndpointBuilder).build();

                        try {
                                for (Long id : ids) {
                                        strings = artworkEndpoint.getArtWorkName(id).
                                             execute()
                                                        .getItems();
                                        values.addAll(strings);
                                }
                                return values;
                        } catch (Exception e) {

                        }
                        return null;
                }
                @Override
                protected void onPostExecute(List<String> result) {
                        if (!values.isEmpty() && values != null)
                                populateFavoritesList(true);
                }

        }

        private class getFavoritesTask extends AsyncTask<Void, Void, Void> {

                @Override
                protected Void doInBackground(Void... params) {
                        Useraccountendpoint.Builder userAccountEndpointBuilder =
                             new Useraccountendpoint.Builder(
                                        AndroidHttp.newCompatibleTransport(), new
                                             JacksonFactory(),
                                        new HttpRequestInitializer() {

                                                @Override
                                                public void initialize(HttpRequest
                                                     httpRequest)
                                                                throws IOException {

                                                }
                                        });

                        Useraccountendpoint useraccountendpoint =
                             CloudEndpointUtils
                                        .updateBuilder(userAccountEndpointBuilder).
                                             build();

                        try {
                                ids = useraccountendpoint.getFavorites(accountName).
                                     execute()
                                                .getItems();

                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        return null;
                }

                @SuppressWarnings("unchecked")
                @Override
                protected void onPostExecute(Void result) {
                        if (!ids.isEmpty() && ids != null)
                                new getFavoriteNamesTask().execute(ids);
                }

        }
}
```

```
package com.example.pubart;

import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.example.pubart.artworkrouteendpoint.Artworkrouteendpoint;
import com.example.pubart.artworkrouteendpoint.model.ArtworkRoute;
import com.example.pubart.helpers.MyBaseActivity;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson.JacksonFactory;

public class ListRoutesActivity extends MyBaseActivity {

       ArrayList<RouteListItem> listOfRoutes;
       ListView listViewItems;

       @Override
       protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_list_routes);
              listOfRoutes = new ArrayList<ListRoutesActivity.RouteListItem>();
              listViewItems = new ListView(this);
              listViewItems = (ListView) findViewById(R.id.lvRouteOverview);

              new getArtworkRoutesTask().execute();
       }

       class RouteListArrayAdapter extends ArrayAdapter<RouteListItem> {
              Context mContext;
              int layoutResourceId;
              ArrayList<RouteListItem> data = new ArrayList<ListRoutesActivity.
                   RouteListItem>();

              public RouteListArrayAdapter(Context mContext, int
                   layoutResourceId,
                             ArrayList<RouteListItem> data) {
                     super(mContext, layoutResourceId, data);

                     this.mContext = mContext;
                     this.layoutResourceId = layoutResourceId;
                     this.data = data;
              }

              @Override
```

```
        public View getView(int position, View convertView, ViewGroup
             parent) {
            if (convertView == null) {
                   LayoutInflater inflater = LayoutInflater.from(
                        mContext);
                   convertView = inflater.inflate(layoutResourceId,
                        parent, false);
            }

            RouteListItem routeListItem = data.get(position);

            TextView tvName = (TextView) convertView
                           .findViewById(R.id.tvRouteName);
            TextView tvDescription = (TextView) convertView
                           .findViewById(R.id.tvRouteDescription);
            tvName.setText(routeListItem.routeName);
            tvName.setTag(routeListItem.routeId);
            tvDescription.setText(routeListItem.routeDescription);

            return convertView;
        }
}

class onItemClickListenerRouteListItem implements OnItemClickListener {

        @Override
        public void onItemClick(AdapterView<?> parent, View view, int
             position,
                   long id) {

            Context context = view.getContext();

            TextView tvName = (TextView) view.findViewById(R.id.
                 tvRouteName);
            TextView tvDescription = (TextView) view
                           .findViewById(R.id.tvRouteDescription);

            String tvNameText = tvName.getText().toString();
            String tvDescriptionText = tvDescription.getText().toString
                 ();
            Long routeId = Long.parseLong(tvName.getTag().toString());

            Intent startRouteMapIntent = new Intent(
                 getApplicationContext(),
                           RouteMapActivity.class);
            startRouteMapIntent.putExtra("ROUTE_ID", routeId);
            startActivity(startRouteMapIntent);


        }

}

class RouteListItem {
        public String routeName;
        public String routeDescription;
        public Long routeId;
```

```
            public RouteListItem(String name, String description, Long id) {
                    this.routeName = name;
                    this.routeDescription = description;
                    this.routeId = id;
            }
    }

    private class getArtworkRoutesTask extends
                    AsyncTask<Void, Void, List<ArtworkRoute>> {

            List<ArtworkRoute> listOfRoutesFromServer = new ArrayList<
                 ArtworkRoute>();

            @Override
            protected List<ArtworkRoute> doInBackground(Void... params) {
                    Artworkrouteendpoint.Builder artworkrouteEndpointBuilder =
                         new Artworkrouteendpoint.Builder(
                                    AndroidHttp.newCompatibleTransport(), new
                                         JacksonFactory(),
                                    new HttpRequestInitializer() {

                                            @Override
                                            public void initialize(HttpRequest
                                                 httpRequest) {
                                                    // TODO Auto-generated method
                                                         stub

                                            }
                                    });

                    Artworkrouteendpoint artworkrouteEndpoint =
                         CloudEndpointUtils
                                    .updateBuilder(artworkrouteEndpointBuilder).
                                         build();

                    try {
                            listOfRoutesFromServer = artworkrouteEndpoint
                                            .listArtworkRoute().execute().
                                                 getItems();
                            return listOfRoutesFromServer;
                    } catch (Exception e) {
                            e.printStackTrace();
                    }

                    return null;
            }

            @Override
            protected void onPostExecute(List<ArtworkRoute> result) {
                    for (ArtworkRoute route : result) {
                            RouteListItem item = new RouteListItem(route.getName
                                 (),
                                            route.getDescription(), route.getId()
                                                 );
                            listOfRoutes.add(item);
                    }
```

```
                        RouteListArrayAdapter adapter = new RouteListArrayAdapter(
                                        getApplicationContext(), R.layout.
                                             listview_artworkroutes,
                                        listOfRoutes);

                        listViewItems.setAdapter(adapter);
                        listViewItems
                                        .setOnItemClickListener(new
                                             onItemClickListenerRouteListItem());
                }
        }

}
```

```
package com.example.pubart;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import android.accounts.AccountManager;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

import com.example.pubart.artworkendpoint.Artworkendpoint;
import com.example.pubart.artworkendpoint.model.ArtWork;
import com.example.pubart.artworkendpoint.model.ArtWorkCollection;
import com.example.pubart.helpers.GeoLocationHelper;
import com.example.pubart.helpers.MyBaseActivity;
import com.example.pubart.useraccountendpoint.Useraccountendpoint;
import com.example.pubart.useraccountendpoint.model.UserAccount;
import com.google.android.gms.common.AccountPicker;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.OnInfoWindowClickListener;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson.JacksonFactory;

public class MapActivity extends MyBaseActivity {
        private static final String TAG = "MapActivity";
        public static final String PREFS_NAME = "PubArtPreferences";
        private static final int REQUEST_ACCOUNT_PICKER = 2;

        private GoogleMap map;
        private GeoLocationHelper geoLocationHelper = new GeoLocationHelper();
        private ArrayList<ArtworkEntity> downloadedArtworks;
```

```
	private Map<String, Long> markers;
	private SharedPreferences preferences;
	private String accountName;
	private Integer insertUserResultCode;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		// Set contentview to XML file
		setContentView(R.layout.activity_map);

		preferences = getSharedPreferences(PREFS_NAME, 0);
		if (preferences.getString("ACCOUNT_ID", "None").equals("None")) {
			chooseAccount();
		} else {
			accountName = preferences.getString("ACCOUNT_ID", "No
			     account");
		}

		if (downloadedArtworks == null || downloadedArtworks.isEmpty()) {
			downloadedArtworks = new ArrayList<ArtworkEntity>();
		}
		if (markers == null || markers.isEmpty()) {
			markers = new HashMap<String, Long>();
		}

		geoLocationHelper.startRetrievingLocation(this);

		// Create map
		map = ((MapFragment) getFragmentManager().findFragmentById(R.id.
		     map))
				.getMap();

		map.setOnInfoWindowClickListener(new OnInfoWindowClickListener() {

			@Override
			public void onInfoWindowClick(Marker marker) {

				Intent startArtworkInformationIntent = new Intent(
						getApplicationContext(),
						ArtworkInformationActivity.class);
				startArtworkInformationIntent.putExtra("ARTWORK_ID",
						markers.get(marker.getId()));
				startArtworkInformationIntent.putExtra("ACCOUNT_ID",
						accountName);
				startActivity(startArtworkInformationIntent);

			}
		});

		new getArtworksTask().execute();

	}

	public String getAccountName() {
		if (!accountName.isEmpty() || accountName != null) {
			return accountName;
```

```
                } else {
                        return "No account provided";
                }
        }

        public void chooseAccount() {
                Intent intent = AccountPicker.newChooseAccountIntent(null, null,
                                new String[] { "com.google" }, true, null, null,
                                     null, null);
                startActivityForResult(intent, REQUEST_ACCOUNT_PICKER);
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent
             data) {
                if (requestCode == REQUEST_ACCOUNT_PICKER && resultCode ==
                     RESULT_OK) {
                        accountName = data.getExtras().getString(
                                        AccountManager.KEY_ACCOUNT_NAME);
                        if (accountName != null) {
                                new insertUserTask().execute(accountName);
                                SharedPreferences.Editor editor = preferences.edit()
                                     ;
                                editor.putString("ACCOUNT_ID", accountName);
                                editor.commit();

                        }
                }
        }

        private class getArtworksTask extends AsyncTask<Void, Void, List<ArtWork
             >> {

                @Override
                protected void onPreExecute() {
                        Toast.makeText(getApplicationContext(), "Retrieving
                             artworks",
                                        Toast.LENGTH_LONG).show();
                }

                @Override
                protected void onPostExecute(List<ArtWork> artworks) {
                        if (artworks == null || artworks.isEmpty()) {
                                Toast.makeText(getApplicationContext(),
                                                "Failed to load artworks", Toast.
                                                     LENGTH_SHORT).show();
                        } else {
                                for (ArtWork artworkInList : artworks) {
                                        ArtworkEntity downloadedArtwork = new
                                             ArtworkEntity(
                                                        artworkInList.getId(),
                                                        artworkInList.getName(),
                                                        artworkInList.
                                                             getTextualDescription(),
                                                        artworkInList.
                                                             getTextualBackground(),
                                    artworkInList.getYear(),
                                        artworkInList.getArtist(),
                                    artworkInList.getCategory(),
                                    artworkInList.getLatitude(),
                                    artworkInList.getLongitude(),
                                    artworkInList.getAverageRating
                                        ());
                        downloadedArtworks.add(downloadedArtwork);
                        Marker marker = map.addMarker(new
                            MarkerOptions()
                                    .position(
                                                new LatLng(
                                                    downloadedArtwork
                                                    .
                                                    getLatitude
                                                    (),
                                                            downloadedArtwork
                                                                .
                                                                getLongitude
                                                                ()
                                                                )
                                                                )


                                    .title(downloadedArtwork.
                                        getName())
                                    .snippet(downloadedArtwork.
                                        getShortDescription()));
                        markers.put(marker.getId(), artworkInList.
                            getId());
                }
            }

        }

        @Override
        protected List<ArtWork> doInBackground(Void... parms) {

            List<ArtWork> artworks = new ArrayList<ArtWork>();

            Artworkendpoint.Builder artworkEndpointBuilder = new
                Artworkendpoint.Builder(
                        AndroidHttp.newCompatibleTransport(), new
                            JacksonFactory(),
                        new HttpRequestInitializer() {

                            @Override
                            public void initialize(HttpRequest
                                httpRequest) {

                            }
                        });
            Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                updateBuilder(
                        artworkEndpointBuilder).build();

            try {
```

```
                        artworks = artworkEndpoint.listArtWork().execute().
                            getItems();
                        Log.d(TAG, "getting artwork");
                        return artworks;
                } catch (Exception exception) {
                        exception.printStackTrace();
                        Log.e(TAG, "IOException on doInBackground",
                            exception);
                }

                return null;
        }

}

private class insertUserTask extends AsyncTask<String, Void, Void> {

        protected void onPostExecute() {
                if (insertUserResultCode == 1) {
                        Toast.makeText(getApplicationContext(),
                                        "Username saved successfully", Toast.
                                            LENGTH_LONG)
                                        .show();
                } else if (insertUserResultCode == 0) {
                        Toast.makeText(getApplicationContext(), "Username
                            not saved",
                                        Toast.LENGTH_LONG).show();
                } else {
                        Toast.makeText(getApplicationContext(),
                                        "Something went wrong, username not
                                            saved",
                                        Toast.LENGTH_LONG).show();
                }
        }

        @Override
        protected void onPreExecute() {
                Toast.makeText(getApplicationContext(),
                                "Account not found, creating new", Toast.
                                    LENGTH_LONG)
                                .show();
        }

        @Override
        protected Void doInBackground(String... params) {

                Useraccountendpoint.Builder userEndpointBuilder = new
                    Useraccountendpoint.Builder(
                                AndroidHttp.newCompatibleTransport(), new
                                    JacksonFactory(),
                                new HttpRequestInitializer() {

                                        @Override
                                        public void initialize(HttpRequest
                                            httpRequest) {

                                        }
```

```
                              });
                        Useraccountendpoint userEndPoint = CloudEndpointUtils
                              .updateBuilder(userEndpointBuilder).build();
                        try {
                              UserAccount userAccount = new UserAccount();
                              userAccount.setEmail(accountName);
                              UserAccount result = userEndPoint
                                          .insertUserAccount(userAccount).
                                                execute();
                              insertUserResultCode = 1;
                        } catch (Exception E) {
                              insertUserResultCode = 0;
                              Log.e(TAG, "IO exception on doInBackground", E);
                        }
                        return null;
                  }
            }
}
```

```
package com.example.pubart;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import com.example.pubart.artworkendpoint.Artworkendpoint;
import com.example.pubart.artworkendpoint.model.ArtWork;
import com.example.pubart.artworkrouteendpoint.Artworkrouteendpoint;
import com.example.pubart.artworkrouteendpoint.model.ArtworkRoute;
import com.example.pubart.helpers.GeoLocationHelper;
import com.example.pubart.helpers.MyBaseActivity;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.GoogleMap.OnInfoWindowClickListener;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.PolylineOptions;
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestInitializer;
import com.google.api.client.json.jackson.JacksonFactory;

import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.view.Menu;

public class RouteMapActivity extends MyBaseActivity {

      private GoogleMap map;
      private GeoLocationHelper geoLocationHelper = new GeoLocationHelper();
      private Map<String, Long> markers;
      private String accountName;
```

```
        private SharedPreferences preferences;
        private Long routeId;
        ArtworkRoute artworkRoute;
        PolylineOptions lineOptions;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_route_map);

                preferences = getSharedPreferences(PREFS_NAME, 0);
                if (preferences.getString("ACCOUNT_ID", "None").equals("None")) {

                } else {
                        accountName = preferences.getString("ACCOUNT_ID", "No
                             account");
                }

                if (markers == null || markers.isEmpty()) {
                        markers = new HashMap<String, Long>();
                }

                Bundle extras = getIntent().getExtras();
                if (extras != null) {
                        routeId = extras.getLong("ROUTE_ID");
                }

                lineOptions = new PolylineOptions();

                geoLocationHelper.startRetrievingLocation(this);

                // Create map
                map = ((MapFragment) getFragmentManager().findFragmentById(R.id.
                     map))
                                .getMap();

                map.setOnInfoWindowClickListener(new OnInfoWindowClickListener() {

                        @Override
                        public void onInfoWindowClick(Marker marker) {

                                Intent startArtworkInformationIntent = new Intent(
                                                getApplicationContext(),
                                                ArtworkInformationActivity.class);
                                startArtworkInformationIntent.putExtra("ARTWORK_ID",
                                                markers.get(marker.getId()));
                                startArtworkInformationIntent.putExtra("ACCOUNT_ID",
                                                accountName);
                                startActivity(startArtworkInformationIntent);

                        }
                });

                new getRouteTask().execute(routeId);
        }

        private class getRouteTask extends AsyncTask<Long, Void, ArtworkRoute> {
```

```
        @Override
        protected ArtworkRoute doInBackground(Long... params) {

                Artworkrouteendpoint.Builder artworkRouteEndpointBuilder =
                     new Artworkrouteendpoint.Builder(
                                AndroidHttp.newCompatibleTransport(), new
                                     JacksonFactory(),
                                new HttpRequestInitializer() {

                                        @Override
                                        public void initialize(HttpRequest
                                             httpRequest) {

                                        }
                                });

                Artworkrouteendpoint artworkRouteEndpoint =
                     CloudEndpointUtils
                                .updateBuilder(artworkRouteEndpointBuilder).
                                     build();

                try {
                        artworkRoute = artworkRouteEndpoint.getArtworkRoute(
                             params[0])
                                        .execute();
                } catch (IOException ioe) {
                        ioe.printStackTrace();
                }

                return artworkRoute;

        }

        @Override
        protected void onPostExecute(ArtworkRoute result) {
                for (Long artworkID : artworkRoute.getArtworks()) {
                        new getArtworkTask().execute(artworkID);
                }
        }

}

private class getArtworkTask extends AsyncTask<Long, Void, ArtWork> {

        ArtWork thisArtwork;

        @Override
        protected ArtWork doInBackground(Long... params) {
                Artworkendpoint.Builder artworkEndpointBuilder = new
                     Artworkendpoint.Builder(
                                AndroidHttp.newCompatibleTransport(), new
                                     JacksonFactory(),
                                new HttpRequestInitializer() {

                                        @Override
```

```
                                    public void initialize(HttpRequest
                                         httpRequest) {

                                    }
                              });
                  Artworkendpoint artworkEndpoint = CloudEndpointUtils.
                       updateBuilder(
                              artworkEndpointBuilder).build();

                  try {
                        thisArtwork = artworkEndpoint.getArtWork(params[0]).
                             execute();
                  } catch (IOException ioe) {
                        ioe.printStackTrace();
                  }

                  return thisArtwork;
            }

            @Override
            protected void onPostExecute(ArtWork result) {
                  Marker marker = map.addMarker(new MarkerOptions()
                              .position(
                                          new LatLng(result.getLatitude
                                               (), result
                                                       .getLongitude())
                                                            ).title(
                                                            result.
                                                            getName())
                              .snippet(result.getTextualDescription()));
                  markers.put(marker.getId(), result.getId());
                  lineOptions.add(new LatLng(result.getLatitude(), result.
                       getLongitude()));
                  if(lineOptions.getPoints().size() > 1) {
                        map.addPolyline(lineOptions);
                  }
            }

      }

}
```

B.2 App Engine

B.2.1 Datastore

```
package com.example.pubart;

import java.util.ArrayList;
import java.util.List;

import com.googlecode.objectify.annotation.Embed;
import com.googlecode.objectify.annotation.Entity;
```

```java
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

@Entity
public class ArtWork {

	@Id
	private Long id;

	@Index
	private String name;
	private String artist;
	private String year;
	private String category;
	private String textualDescription;
	private String textualBackground;
	private Double latitude;
	private Double longitude;
	private Integer averageRating;

	@Embed
	public static class Rating {

		@Index
		private String userEmail;
		private Integer userRating;

		public void setUserEmail(String userEmail) {
			this.userEmail = userEmail;
		}

		public String getUserEmail() {
			return userEmail;
		}

		public void setRating(Integer rating) {
			this.userRating = rating;
		}

		public Integer getRating() {
			return userRating;
		}
	}

	@Index
	private List<Rating> ratings = new ArrayList<Rating>();

	public ArtWork() {
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getArtist() {
        return artist;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public String getTextualDescription() {
        return textualDescription;
    }

    public void setTextualDescription(String textualDescription) {
        this.textualDescription = textualDescription;
    }

    public String getTextualBackground() {
        return textualBackground;
    }

    public void setTextualBackground(String textualBackground) {
        this.textualBackground = textualBackground;
    }

    public Double getLatitude() {
        return latitude;
    }

    public void setLatitude(Double latitude) {
        this.latitude = latitude;
    }
```

```
        public Double getLongitude() {
                return longitude;
        }

        public void setLongitude(Double longitude) {
                this.longitude = longitude;
        }

        public Integer getAverageRating() {
                return averageRating;
        }

        public void setAverageRating(Integer averageRating) {
                this.averageRating = averageRating;
        }

        public List<Rating> getRatings() {
                return ratings;
        }

        public void setRatings(List<Rating> ratings) {
                this.ratings = ratings;
        }

        public void addRating(Rating rating) {
                ratings.add(rating);
                int sum = 0;
                for(Rating ratingEntry : ratings) {
                        sum += ratingEntry.getRating();
                }
                averageRating = sum / ratings.size();
        }

        public void editRating(String userName, Integer rating) {
                for (Rating oldRating : ratings) {
                        if (oldRating.userEmail.equals(userName)) {
                                oldRating.userRating = rating;
                                break;
                        }
                }
        }

}
```

```
package com.example.pubart;

import java.util.ArrayList;
import java.util.List;

import com.googlecode.objectify.annotation.Embed;
import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

@Entity
public class ArtworkRoute {
```

```
@Id
private Long id;
private String name;
private List<Long> artworks = new ArrayList<Long>();
private String description;
@Index
private Integer averageRating;
private String background;

@Embed
public static class Rating {
	private String userEmail;

	private Integer userRating;

	public Rating() {

	}

	public Rating(String userEmail, Integer userRating) {
		this.userEmail = userEmail;
		this.userRating = userRating;
	}

	public void setUserEmail(String userEmail) {
		this.userEmail = userEmail;
	}

	public String getUserEmail() {
		return userEmail;
	}

	public void setRating(Integer rating) {
		this.userRating = rating;
	}

	public Integer getRating() {
		return userRating;
	}
}

private List<Rating> ratings = new ArrayList<Rating>();

public ArtworkRoute() {}

public Long getId() {
	return id;
}

public void setId(Long id) {
	this.id = id;
}

public String getName() {
	return name;
}
```

```
	public void setName(String name) {
		this.name = name;
	}

	public List<Long> getArtworks() {
		return artworks;
	}

	public void setArtworks(List<Long> artworks) {
		this.artworks = artworks;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public Integer getAverageRating() {
		return averageRating;
	}

	public void setAverageRating(Integer averageRating) {
		this.averageRating = averageRating;
	}

	public List<Rating> getRatings() {
		return ratings;
	}

	public void setRatings(List<Rating> ratings) {
		this.ratings = ratings;
	}

	public void setBackground(String background) {
		this.background = background;
	}

	public String getBackground() {
		return background;
	}


}
```

```
package com.example.pubart;

import java.util.ArrayList;
import java.util.List;

import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;
```

```
@Entity
public class UserAccount {

	@Id
	private String email;
	private List<Long> favorites = new ArrayList<Long>();

	public UserAccount() {}

	public UserAccount(String email) {
		this.email = email;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public List<Long> getFavorites() {
		return favorites;
	}

	public void setFavorites(List<Long> favorites) {
		this.favorites = favorites;
	}

	public void addFavorite(Long id) {
		favorites.add(id);
	}




}
```

B.2.2 Endpoints

```
package com.example.pubart;

import static com.googlecode.objectify.ObjectifyService.ofy;

import java.util.ArrayList;
import java.util.List;

import javax.inject.Named;

import com.example.pubart.ArtWork.Rating;
import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;
```

```
@Api(name = "artworkendpoint", namespace = @ApiNamespace(ownerDomain = "example.
     com", ownerName = "example.com", packagePath = "pubart"))
public class ArtWorkEndpoint {

       /**
        * This method lists all the entities inserted in datastore. It uses HTTP
        * GET method and paging support.
        *
        * @return A List of artworks containing the list of all entities
        *
        */
       @ApiMethod(name = "listArtWork", httpMethod = "GET")
       public List<ArtWork> listArtWork() {

              List<ArtWork> listOfArtWorks = ofy().load().type(ArtWork.class).
                   list();
              return listOfArtWorks;

       }

       @ApiMethod(name = "getTopTenArtworks", path="get_top_ten_artworks",
            httpMethod = "GET")
       public List<ArtWork> getTopTenArtworks() {
              List<ArtWork> topTenArtworks = ofy().load().type(ArtWork.class).
                   order("averageRating").list();
              return topTenArtworks;
       }

       /**
        * This method gets the entity having primary key id. It uses HTTP GET
        * method.
        *
        * @param id
        *           the primary key of the java bean.
        * @return The entity with primary key id.
        */
       @ApiMethod(name = "getArtWork")
       public ArtWork getArtWork(@Named("id") Long id) {
              ArtWork artwork = ofy().load().type(ArtWork.class).id(id).get();
              return artwork;
       }

       /**
        * This inserts a new entity into App Engine datastore. If the entity
        * already exists in the datastore, an exception is thrown. It uses HTTP
        * POST method.
        *
        * @param artwork
        *           the entity to be inserted.
        * @return The inserted entity.
        */
       @ApiMethod(name = "insertArtWork")
       public ArtWork insertArtWork(ArtWork artwork) {
              ofy().save().entity(artwork).now();
              return artwork;
       }
```

```
    /**
     * This method is used for updating an existing entity. If the entity does
     * not exist in the datastore, an exception is thrown. It uses HTTP PUT
     * method.
     *
     * @param artwork
     *          the entity to be updated.
     * @return The updated entity.
     */
    @ApiMethod(name = "updateArtWork")
    public ArtWork updateArtWork(ArtWork artwork) {
            ArtWork artworkToUpdate = new ArtWork();
            artworkToUpdate.setId(artwork.getId());
            artworkToUpdate.setName(artwork.getName());
            artworkToUpdate.setArtist(artwork.getArtist());
            artworkToUpdate.setYear(artwork.getYear());
            artworkToUpdate.setCategory(artwork.getCategory());
            artworkToUpdate.setTextualDescription(artwork.
                 getTextualDescription());
            artworkToUpdate.setTextualBackground(artwork.getTextualBackground
                 ());
            artworkToUpdate.setLatitude(artwork.getLatitude());
            artworkToUpdate.setLongitude(artwork.getLongitude());
            artworkToUpdate.setAverageRating(artwork.getAverageRating());
            artworkToUpdate.setRatings(artwork.getRatings());
            ofy().save().entity(artworkToUpdate).now();
            return artworkToUpdate;
    }

    /**
     * This method is used to add a rating to an artwork entity.
     *
     * @param artwork
     *          the entity to be updated
     * @return the updated artwork
     */

    @ApiMethod(name = "insertUserRating", httpMethod = "PUT", path = "
         insertRating")
    public ArtWork insertUserRating(ArtWork artwork,
                    @Named("userName") String userName, @Named("rating")
                         Integer rating) {
            boolean exists = ((ofy().load().type(ArtWork.class)
                            .filter("ratings.userEmail =", userName).count()) >
                                 0);
            if (exists) {
                    artwork.editRating(userName, rating);
            } else {
                    Rating ratingToAdd = new Rating();
                    ratingToAdd.setUserEmail(userName);
                    ratingToAdd.setRating(rating);
                    artwork.addRating(ratingToAdd);
            }
            ofy().save().entity(artwork).now();
            return artwork;
    }
```

```
/**
 * This method is used to get a rating for a user for an artwork
 *
 * @param artwork
 *          the entity to get rating for
 * @param userName
 *          the user to get rating for
 * @return the rating, or -1 if username or rating not found.
 */

@ApiMethod(name = "getUserRatingForArtwork", httpMethod = "POST", path =
     "getUserRatingForArtwork")
public List<Integer> getUserRatingForArtwork(@Named("id") Long id,
              @Named("userName") String userName) {
       List<Integer> returnList = new ArrayList<>();
       ArtWork artwork = ofy().load().type(ArtWork.class).id(id).get();
       ArrayList<Rating> ratings = new ArrayList<>();
       ratings.addAll(artwork.getRatings());
       for (Rating rating : ratings) {
              if (rating.getUserEmail().equals(userName)) {
                     returnList.add(rating.getRating());
              }
       }
       if (returnList.isEmpty()) {
              returnList.add(3);
       }
       return returnList;

}

/**
 * This method removes the entity with primary key id. It uses HTTP DELETE
 * method.
 *
 * @param id
 *          the primary key of the entity to be deleted.
 */
@ApiMethod(name = "removeArtWork")
public void removeArtWork(@Named("id") Long id) {
       ofy().delete().type(ArtWork.class).id(id).now();
}

/**
 * This method gets the name of the artwork with the primary key id It
     uses
 * HTTP GET
 *
 * @param id
 *          the primary key of the entity
 * @return name the name of the artwork
 */

@ApiMethod(name = "getArtWorkName", httpMethod = "GET", path = "
     getArtWorkName")
public List<String> getArtWorkName(@Named("id") Long id) {
       String name = ofy().load().type(ArtWork.class).id(id).get()
```

```
                              .getName();
                List<String> listOfNames = new ArrayList<>();
                listOfNames.add(name);
                return listOfNames;
        }

}
```

```
package com.example.pubart;

import static com.googlecode.objectify.ObjectifyService.ofy;

import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;

import java.util.List;

import javax.inject.Named;



@Api(name = "artworkrouteendpoint", namespace = @ApiNamespace(ownerDomain = "
     example.com", ownerName = "example.com", packagePath = "pubart"))
public class ArtworkRouteEndpoint {

        /**
         * This method lists all the entities inserted in datastore.
         * It uses HTTP GET method and paging support.
         *
         * @return A CollectionResponse class containing the list of all entities
         * persisted and a cursor to the next page.
         */
        @ApiMethod(name = "listArtworkRoute", path="listArtworkRoute", httpMethod
              = "GET")
        public List<ArtworkRoute> listArtworkRoute() {
                        List<ArtworkRoute> listOfArtworkRoutes = ofy().load().type(
                             ArtworkRoute.class).list();
                        return listOfArtworkRoutes;
        }

        /**
         * This method gets the top 10 rated routes in the datastore
         * It uses HTTP GET method
         *
         * @return A CollectionResponse class containing the list of the top10
             rated routes
         */
        @ApiMethod(name = "listTop10Routes", httpMethod = "GET")
        public List<ArtworkRoute> listTop10Routes() {
                List<ArtworkRoute> listOfTop10Routes = ofy().load().type(
                     ArtworkRoute.class).list();
                return listOfTop10Routes;

        }
```

```
/**
 * This method gets the entity having primary key id. It uses HTTP GET
     method.
 *
 * @param id the primary key of the java bean.
 * @return The entity with primary key id.
 */
@ApiMethod(name = "getArtworkRoute", httpMethod = "GET")
public ArtworkRoute getArtworkRoute(@Named("id") Long id) {
       ArtworkRoute artworkRoute = ofy().load().type(ArtworkRoute.class).
            id(id).get();
       return artworkRoute;
}

/**
 * This inserts a new entity into App Engine datastore. If the entity
     already
 * exists in the datastore, an exception is thrown.
 * It uses HTTP POST method.
 *
 * @param artworkroute the entity to be inserted.
 * @return The inserted entity.
 */
@ApiMethod(name = "insertArtworkRoute")
public ArtworkRoute insertArtworkRoute(ArtworkRoute artworkroute) {
       ofy().save().entity(artworkroute).now();
       return artworkroute;
}

/**
 * This method is used for updating an existing entity. If the entity does
      not
 * exist in the datastore, an exception is thrown.
 * It uses HTTP PUT method.
 *
 * @param artworkroute the entity to be updated.
 * @return The updated entity.
 */
@ApiMethod(name = "updateArtworkRoute")
public ArtworkRoute updateArtworkRoute(ArtworkRoute artworkroute) {
       ArtworkRoute artworkRouteToUpdate = new ArtworkRoute();
       artworkRouteToUpdate.setId(artworkroute.getId());
       artworkRouteToUpdate.setName(artworkroute.getName());
       artworkRouteToUpdate.setDescription(artworkroute.getDescription())
            ;
       artworkRouteToUpdate.setBackground(artworkroute.getBackground());
       artworkRouteToUpdate.setArtworks(artworkroute.getArtworks());
       artworkRouteToUpdate.setAverageRating(artworkroute.
            getAverageRating());
       ofy().save().entity(artworkRouteToUpdate).now();
       return artworkRouteToUpdate;
}

/**
 * This method removes the entity with primary key id.
 * It uses HTTP DELETE method.
 *
```

```
	 * @param id the primary key of the entity to be deleted.
	 */
	@ApiMethod(name = "removeArtworkRoute")
	public void removeArtworkRoute(@Named("id") Long id) {
		ofy().delete().type(ArtworkRoute.class).id(id).now();
	}

	/**
	 * This method gets the top10 rated artworkroutes
	 * It uses HTTP GET method
	 */
	@ApiMethod(name = "getTopTenRatedRoutes", path="get_top_ten_rated_routes"
	     , httpMethod = "GET")
	public List<ArtworkRoute> topRatedRoutes() {
		List<ArtworkRoute> topRatedRoutes = ofy().load().type(ArtworkRoute
		     .class).order("-averageRating").list();
		return topRatedRoutes;
	}

}
```

```
package com.example.pubart;

import static com.googlecode.objectify.ObjectifyService.ofy;

import java.util.ArrayList;
import java.util.List;


import javax.inject.Named;

import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;
import com.googlecode.objectify.Key;


@Api(name = "useraccountendpoint", namespace = @ApiNamespace(ownerDomain = "
     example.com", ownerName = "example.com", packagePath = "pubart"))
public class UserAccountEndpoint {

	/**
	 * This method lists all the entities inserted in datastore.
	 * It uses HTTP GET method and paging support.
	 *
	 * @return A CollectionResponse class containing the list of all entities
	 * persisted and a cursor to the next page.
	 */
	@ApiMethod(name = "listUserAccount", httpMethod = "GET")
	public List<UserAccount> listUserAccounts() {
		List<UserAccount> listOfUserAccounts = ofy().load().type(
		     UserAccount.class).list();
		return listOfUserAccounts;
	}

	/**
```

```
	 * This method gets the entity having primary key id. It uses HTTP GET
	     method.
	 *
	 * @param id the primary key of the java bean.
	 * @return The entity with primary key id.
	 */
	@ApiMethod(name = "getUserAccount")
	public UserAccount getUserAccount(@Named("id") String id) {
		UserAccount userAccount = ofy().load().type(UserAccount.class).id(
		     id).get();
		return userAccount;
	}

	/**
	 * This inserts a new entity into App Engine datastore. If the entity
	     already
	 * exists in the datastore, an exception is thrown.
	 * It uses HTTP POST method.
	 *
	 * @param useraccount the entity to be inserted.
	 * @return The inserted entity.
	 */
	@ApiMethod(name = "insertUserAccount")
	public UserAccount insertUserAccount(UserAccount useraccount) {
		Key<UserAccount> result = ofy().save().entity(useraccount).now();
		return useraccount;
	}

	/**
	 * This method is used for updating an existing entity. If the entity does
	      not
	 * exist in the datastore, an exception is thrown.
	 * It uses HTTP PUT method.
	 *
	 * @param useraccount the entity to be updated.
	 * @return The updated entity.
	 */
	@ApiMethod(name = "updateUserAccount")
	public UserAccount updateUserAccount(UserAccount useraccount) {
		UserAccount userAccountToUpdate = new UserAccount();
		userAccountToUpdate.setEmail(useraccount.getEmail());
		userAccountToUpdate.setFavorites(useraccount.getFavorites());
		Key<UserAccount> result = ofy().save().entity(userAccountToUpdate)
		     .now();
		return userAccountToUpdate;
	}

	/**
	 * This method removes the entity with primary key id.
	 * It uses HTTP DELETE method.
	 *
	 * @param id the primary key of the entity to be deleted.
	 */
	@ApiMethod(name = "removeUserAccount")
	public void removeUserAccount(@Named("id") String id) {
		ofy().delete().type(UserAccount.class).id(id).now();
	}
```

```
        /**
         * This method adds an artwork id to the list of favorites
         * It uses HTTP PUT
         *
         * @param id the id of the artwork to be added
         * @param useraccount the entity to add the artwork id to
         */

        @ApiMethod(name = "addFavorite", httpMethod = "PUT", path = "addFavorite"
             )
        public void addFavorite(@Named("id") Long id, @Named("accountName")
             String accountName) {
               UserAccount userAccount = ofy().load().type(UserAccount.class).id(
                    accountName).get();
               userAccount.addFavorite(id);
               ofy().save().entity(userAccount).now();
        }

        @ApiMethod(name = "getFavorites", httpMethod = "GET", path = "
             getFavorites")
        public List<Long> getFavorites(@Named("id") String id) {
               UserAccount userAccount = ofy().load().type(UserAccount.class).id(
                    id).get();
               List<Long> ids = userAccount.getFavorites();
               return ids;
        }

}
```